Міністерство освіти і науки України
Сумський державний університет
Навчально-науковий інститут бізнес-технологій «УАБС»
Кафедра економічної кібернетики

# КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему «РОЗРОБКА АВТОМАТИЗОВАНОГО РІШЕННЯ ДЛЯ ВИЯВЛЕННЯ ШАХРАЙСЬКИХ ОПЕРАЦІЙ КЛІЄНТІВ БАНКУ»

Виконав студент 2 курсу, групи ЕК.м-61а

Спеціальності 051«Економіка («Економічна кібернетика»)»

Кондраков Андрій Віталійович

Керівник к.е.н., доцент, Яровенко Г.М.

Суми – 2018 рік

# РЕФЕРАТ

кваліфікаційної магістерської роботи на тему

« РОЗРОБКА АВТОМАТИЗОВАНОГО РІШЕННЯ ДЛЯ ВИЯВЛЕННЯ ШАХРАЙСЬКИХ ОПЕРАЦІЙ КЛІЄНТІВ БАНКУ»

студента Кондракова Андрія Віталійовича

Актуальність кваліфікаційної магістерської роботи зумовлена великою кількістю шахрайських операцій, темпом зростання цієї цифри та негативними наслідками, як для банку, так і для клієнтів.

Мета кваліфікаційної магістерської роботи полягає у розробці автоматизованого рішення для виявлення шахрайських операцій клієнтів банку. Об'єктом є взаємовідносини між банками та їх клієнтами в процесі здійснення банківських операцій. Одним із їх різновидів є електронні платежі, які частіше стають об'єктами для незаконного збагачення шахраїв.

Предметом дослідження виступають інформаційні технології та автоматизовані рішення для виявлення шахрайських операцій клієнтів банку.

Методи дослідження: аналіз і синтез, дедукція, абстрагування, конкретизація, аргументація, порівняння, класифікація та узагальнення.

Інформаційно-фактологічну базу дослідження сформували: звіти Національного банку України щодо статистики шахрайства в банківській системі країни; положення Національного банку України про системи електронних платежів; положення Департаменту фінансового моніторингу України щодо проведення банківських операцій та електронних платежів; документації для розробників про основні методи фреймворку Ruby on Rails; документації для розробників про основні методи бібліотеки React; документації для розробників з використання серверу WebPack.

В результаті роботи сформульовані наступні висновки: існуючі системи електронних платежів дозволяють здійснювати шахрайські операції клієнтів банку.

Розроблену систему можна впровадити в існуюччі системи електронних платежів, що знизить кількість шахрайских операцій клієнтів банку.

Ключові слова: система електронних платежів, шахрайство, шахрайська операція, банківська операція.

Результати проведеної роботи є частиною проекту, розробленого ТОВ “РЕЗОНАНС.НЕТ” на замовлення банківської установи.

Основний зміст кваліфікаційної магістерської роботи викладено на 133 сторінці, зокрема список використаних джерел із 33 найменувань, розміщений на 4 сторінках. Робота містить 9 таблиць, 46 рисунків та 5 формул.

Рік виконання кваліфікаційної роботи  2017 – 2018.

Рік захисту роботи 2018.

Міністерство освіти і науки України

Сумський державний університет

Навчально-науковий інститут бізнес-технологій «УАБС»

Кафедра економічної кібернетики

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.е.н., доцент
_____ О.В. Кузьменко
«__» _____2017 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ МАГІСТЕРСЬКУ РОБОТУ
(спеціальність 051 «Економіка («Економічна кібернетика»)»)
студенту 2 курсу, групи ЕКм-61а

Кондракова Андрія Віталійовича
(прізвище, ім'я, по батькові студента)

1. Тема роботи Розробка автоматизованого рішення для виявлення шахрайських операцій клієнтів банку
затверджена наказом по університету від «08» грудня 2017 року № 2748-III

2. Термін подання студентом закінченої роботи «  » _____ 2018 року

3. Мета кваліфікаційної роботи полягає у розробці автоматизованого рішення для виявлення шахрайських операцій клієнтів банку.

4. Об'єкт дослідження: взаємовідносини між банками та їх клієнтами в процесі здійснення банківських операцій.

5. Предмет дослідження: інформаційні технології та автоматизовані рішення для виявлення шахрайських операцій клієнтів банку.

6. Кваліфікаційна робота виконується на матеріалах: нормативно-довідкова та рекомендаційна документація Національного банку України, Дер ймен фінансового моніторингу України.

7. Орієнтовний план кваліфікаційної роботи, терміни подання розділів керівникові та зміст завдань для виконання поставленої мети

Розділ 1 ДОСЛІДЖЕННЯ МЕХАНІЗМУ РЕАЛІЗАЦІЇ ЕЛЕКТРОННИХ ПЛАТЕЖІВ ТА ПЕРЕШКОДЖАННЯ ШАХРАЙСЬКИМ ФІНАНСОВИМ ОПЕРАЦІЯМ – 13 листопада 2017 р.

(назва – термін подання)

У розділі 1 розкрити сутність банківських операцій як об'єкту шахрайства, проаналізувати існуючі системи електронних платежів, як інструменту здійснення банківськиїх операцій, розробити постановку задачі для розробки автоматизованого рішення виявлення шахрайських операцій

(зміст конкретних завдань до розділу, які повинен виконати студент)

Розділ 2 ПРОЕКТУВАННЯ ПРОТОТИПУ ДОДАТКУ АВТОМАТИЗАЦІЇ ВИЯВЛЕННЯ ШАХРАЙСЬКИХ ОПЕРАЦІЙ – 12 грудня 2017 р.

(назва – термін подання)

У розділі 2 здійснити вибір технологій для розробки програмного додатку, спроектувати структуру бази даних та архітектуру додатку, розробити проект інтерфейсу додатку

(зміст конкретних завдань до розділу, які має виконати студент)

Розділ 3 РЕАЛІЗАЦІЯ ПРОТОТИПУ ДОДАТКУ АВТОМАТИЗАЦІЇ ВИЯВЛЕННЯ ШАХРАЙСЬКИХ ОПЕРАЦІЙ – 4 січня 2018 р.

(назва – термін подання)

У розділі 3 здійснити розробку серверної частини програмного додатку автоматизації виявлення шахрайських операцій, розробку клієнтського інтерфейсу додатку та оцінити очікувані ефекти від впровадження системи

(зміст конкретних завдань до розділу, які повинен виконати студент)

8. Консультації з роботи:

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| 1 | Доц. Яровенко Г.М. | | |
| 2 | Доц. Яровенко Г.М. | | |
| 3 | Доц. Яровенко Г.М. | | |

9. Дата видачі завдання: «___» _____ 20__ року

Керівник кваліфікаційної роботи _____ Г. М. Яровенко

( підпис) (ініціали, прізвище)

Завдання до виконання одержав _____ А. В. Кондраков

(підпис) (ініціали, прізвище)

ЗМІСТ

# ВСТУП

Актуальність даної теми зумовлена великою кількістю шахрайських операцій, темпом зростання цієї цифри та негативними наслідками, як для банку, так і для клієнтів. Це підтверджується даними зі звіту департаменту платіжних систем та іноваційного розвитку Національного банку України - в 2015 році було виявлено 80500 шахрайських операцій, в 2016 їх кількість зросла до 94630. Також підтвердженням є те, що з даною проблемою борються не лише банки, а й держава. Так, Національний банк України зобов'язав банки не покладати на клієнтів відповідальність за спірними операціями з використанням платіжних карт постановою №382 від 6 вересня 2016 року.

Об'єктом дослідження в даній роботі є взаємовідносини між банками та їх клієнтами в процесі здійснення банківських операцій. Одним із їх різновидів є електронні платежі, які частіше стають об'єктами для незаконного збагачення шахраїв.

Предметом дослідження є інформаційні технології та автоматизовані рішення для виявлення шахрайських операцій клієнтів банку.

Метою магістерської кваліфікаційної роботи є розробка автоматизованого рішення для виявлення шахрайських операцій клієнтів банку.

Об'єкт, мета та предмет дослідження зумовили наступні задачі:

- дослідити поняття та механізм роботи банківської операції;
- розглянути системи електронних платежів як інструмент здійснення банківських операцій;
- спроектувати прототип власної платіжної системи та впровадити в ній автоматизоване рішення для виявлення шахрайських операцій;
- спроектувати інтерфейс для користувачів даної платіжної системи;
- розробити серверну частину спроектованої системи;

- розробити клієнтську частину додатку для спроектованої платіжної системи;
- розрахувати економічний ефект від впровадження даного автоматизованого рішення в існуючі платіжні та банківські системи.

Інформаційно-фактологічну базу дослідження сформували: звіти Національного банку України щодо статистики шахрайства в банківській системі країни; положення Національного банку України про системи електронних платежів; положення Департаменту фінансового моніторингу України щодо проведення банківських операцій та електронних платежів; документації для розробників про основні методи фреймворку Ruby on Rails; документації для розробників про основні методи бібліотеки React; документації для розробників з використання серверу WebPack.

Для розробки прототипу для дослідження проблеми було використано наступні технології та додатки: Ruby; Ruby on Rails; Device Auth Token; PostgreSQL; Javascript; Webpack; React; Redux; React Bootstrap; RubyMine; WebStorm; NodeJS.

# РОЗДІЛ 1 ДОСЛІДЖЕННЯ МЕХАНІЗМУ РЕАЛІЗАЦІЇ ЕЛЕКТРОННИХ ПЛАТЕЖІВ ТА ПЕРЕШКОДЖАННЯ ШАХРАЙСЬКИМ ФІНАНСОВИМ ОПЕРАЦІЯМ

## 1.1 Банківські операції як об’єкт шахрайства

Банківські операції, як одні з видів фінансової операції, найчастіше стають об'єктами шахрайства, оскільки їх здійснення пов'язано із різними платіжними засобами.

Згідно з відкритими джерелами “фінансова операція” - це будь-яка операція, пов'язана зі здійсненням або забезпеченням здійснення платежу суб'єктами господарювання, зокрема:

- внесення або зняття депозиту (внеску, вкладу);
- переказ грошей з рахунка на рахунок;
- обмін валюти;
- надання послуг з випуску, купівлі або продажу цінних паперів та інших видів фінансових активів;
- надання або отримання позики або кредиту;
- страхування (перестрахування);
- надання фінансових гарантій та зобов'язань;
- довірче управління портфелем цінних паперів;
- фінансовий лізинг;
- здійснення випуску, обігу, погашення (розповсюдження) державної та іншої грошової лотереї;
- надання послуг з випуску, купівлі, продажу і обслуговування чеків, векселів, платіжних карток, грошових поштових переказів та інших платіжних інструментів;
- відкриття рахунка.

Фінансова операція являє собою угоду, зв'язок, або рух коштів, що здійснюється між покупцем і продавцем, щоб обміняти актив для оплати. Це тягне за собою зміну статусу фінансів двох або більше підприємств або окремих осіб. Покупець і продавець є окремими юридичними особами або об'єктами, що здійснюють обмін цінних предметів, таких, як інформація, товарів, послуг і грошей. Фінансова операція - це двостороння взаємодія: перша частина дає гроші, частина друга отримує товар.

Транзакція - це здійснення закінчених дій стосовно визначеного об'єкта, що переводить цей об'єкт з одного постійного стану в інший.

В економіці, транзакція означає зміну права розпорядження матеріальними благами або послугами, в якій бере участь більш ніж один суб'єкт.

В світі інформаційних технологій, транзакція відіграє важливу роль в базах даних, є логічною одиницею і зобов'язана відповідати принципам ACID (Atomicity, Consistency, Isolation, Durability). Гарантує збереження цілісності бази даних.

В банківській системі, це операція, яка полягає в переведенні коштів з одного рахунку на інший.

Для подальшого дослідження треба визначитись з терміном “шархайство”.

Одним із першоджерел визначення цього терміна є Акт Ради ЄС від 26 липня 1995 року, яким визначено Конвенцію щодо захисту фінансових інтересів Європейських співтовариств. Цей документ надає такі визначення терміна "шахрайство":

- будь яка дія чи навмисне упущення, що включає використання чи представлення неправдивих, неточних чи неповних відомостей або документів, що має наслідком незаконне за володіння, привласнення, розтрата чи неправомірне збереження коштів; неповідомлення інформації в порушення конкретного зобов'язання, що призводить до такого самого наслідку; використання коштів на цілі, відмінні від тих, на які вони видавалися.

Поміж тим, в українському правовому середовищі термін "шахрайство" визначений в дещо вужчому значенні. Так, Кримінальним кодексом України (ККУ) визначено як загальне поняття шахрайства (ст. 190 ККУ), так і поняття його окремого різновиду — шахрайство з фінансовими ресурсами (ст. 222 ККУ).

У статті 190 ККУ поняття "шахрайство" подається в контексті заволодіння чужим майном або придбання права на майно шляхом обману чи зловживання довірою (при цьому необхідно зважити на те, що під майном в національному законодавстві розуміються і кошти). Згідно зі статтею 222 ККУ до поняття "шахрайство з фінансовими ресурсами" віднесено надання громадянином підприємцем або громадянином — засновником (учасником) або службовою особою суб'єкта господарської діяльності неправдивої інформації органам державної влади, органам влади АР Крим чи органам місцевого самоврядування, банкам або іншим кредиторам з метою одержання субсидій, субвенцій, дотацій, кредитів чи пільг по сплаті податків.

Отже, роблячи висновки з перерахованих вище термінів, можна визначити термін - “шахрайська фінансова операція” - це угода, зв'язок, або рух коштів, що здійснюється між двома особами(фізичними або юридичними), в результаті чого одна особа заволодіває активами іншої незаконно.

В сучасній банківській системі існує багато критеріїв для того, щоб вважати фінансову операцію шахрайською. Згідно з офіційними даними Департаменту фінансового моніторингу ми маємо основний перелік критеріїв для виявлення клієнтів, фінансові операції яких містять ознаки фіктивності.

Таблиця 1.1 - Перелік критеріїв для виявлення клієнтів, фінансові операції яких містять ознаки фіктивності

| N | Назва ознаки | Зміст ознаки |
|---|---|---|
| 1. | Основний вид діяльності - оптова торгівля | 1. Основний вид діяльності клієнта згідно з установчими документами - неспеціалізована оптова торгівля, та/або наявний широкий перелік зареєстрованих видів діяльності, які характеризуються окремою специфікою діяльності.<br>2. Клієнт найчастіше не є виробником товару, а виступає посередником, зокрема, здійснює купівлю та продаж різних груп товарів. |
| 2. | Період існування клієнта | Короткий період існування (провадження діяльності), який, як правило, не перевищує податкового звітного періоду (наприклад, квартал), для уникнення перевірок контролюючими органами. |
| 3. | Місцезнаходження | 1. Місцезнаходження клієнта відповідно до відомостей з Єдиного державного реєстру юридичних осіб, фізичних осіб - підприємців та громадських формувань є адресою, яка є місцем реєстрації значної кількості юридичних осіб, та/або такий суб'єкт господарювання не знаходиться за місцем державної реєстрації. |
| 4. | Офшорний статус засновника/учасника | 1. Засновники клієнта є особами з місцем реєстрації у країні, що має офшорний статус, що може свідчити про те, що особи, які є кінцевими бенефіціарними власниками, можуть приховувати свою причетність до цього клієнта, використовуючи послуги, пов'язані з приховуванням причетності до бізнесу (номінального сервісу). |
| 5. | Зміни в діяльності | 1. Часті зміни в установчих документах клієнта, зокрема, пов'язані із зміною його найменування, структури власності та керівників.<br>2. Наявні випадки зміни напрямів іяльності клієнта у разі придбання існуючої компанії (у тому числі |

Продовження таблиці 1.1

| N | Назва ознаки | Зміст ознаки |
|---|---|---|
| | | зі значним обсягом кредиторської заборгованості за товари, роботи, послуги) з метою приховання незаконної діяльності.<br>3. Зміни контрагентів для виплати кредитної заборгованості за товари, роботи, послуги (переуступлення прав вимоги).<br>4. Зміна обслуговуючого банку безпосередньо після зміни власника та/або керівника. |
| 6. | Невідповідність ресурсів обсягам здійснюваної діяльності | 1. У штаті клієнта працює одна або декілька осіб.<br>2. Керівником, бухгалтером та засновником клієнта є одна й та сама особа.<br>3. Несформований статутний капітал або мінімальний розмір капіталу, наприклад, статутний фонд клієнта не перевищує 10000 грн.<br>4. Обіг грошових потоків за рахунками клієнта складає мільйони гривень.<br>5. Мають місце випадки проведення за рахунками таких клієнтів фінансових операцій на суму, що складає мільйони гривень з призначенням "оплата за товар/послуги без ПДВ".<br>6. Купівля-продаж цінних паперів, емітент яких включений до списку емітентів, що мають ознаки фіктивності ("сміттєві" цінні папери) тощо. |
| 7. | "Схемні" розрахунки | 1. Кошти за рахунками клієнта після їх зарахування упродовж одного або декількох найближчих днів "транзитом" перераховуються іншим суб'єктам господарювання з різним призначенням платежу, що може свідчити про проведення оплати за товар без фактичної поставки/переміщення товару (безтоварні операції).<br>2. Вхідний і вихідний залишок за рахунком клієнта є мінімальним та/або нульовим за одночасним проведенням операцій на суми, що складають мільйони гривень протягом дня або декількох найближчих днів.<br>При цьому за рахунками такого клієнта за |

Продовження таблиці 1.1

| N | Назва ознаки | Зміст ознаки |
|---|---|---|
| | | короткий період можуть проводитися фінансові операції на суму, що становить мільйони гривень як оплата за різні види товарів, послуг, в тому числі за договорами переуступки боргу/відступлення прав вимоги, фінансової допомоги тощо. |
| 8. | Спільний зв'язок діяльності | 1. Клієнт та декілька інших суб'єктів господарювання мають спільну адресу електронної пошти, IP-адресу, поштову адресу та/або номери контактних телефонів.<br>2. Такі суб'єкти господарювання можуть мати спільних керівників та довірену особу, яка проводить фінансові операції в банку.<br>Крім того, засновник/кінцевий бенефіціарний власник/керівник клієнта може бути одночасно засновником/кінцевим бенефіціарним власником/керівником у багатьох інших компаніях, які в тому числі обслуговуються в банку. |
| 9 | Негативна інформація | 1. Наявність стосовно клієнта та/або його керівників/представників/власників щодо кримінальних проваджень з розслідування злочинів у сфері господарської діяльності, отриманої банком від:<br>- правоохоронних органів;<br>- Державної служби фінансового моніторингу України;<br>- Єдиного державного реєстру судових рішень;<br>- засобів масової інформації, в тому числі мережі Інтернет;<br>- юридичних осіб / державних органів. |
| 10. | Підроблені документи та неправдива інформація | 1. Реєстрація клієнта на підставну (неіснуючу) особу.<br>2. За підробленими, втраченими або викраденими документами, документами померлих осіб.<br>3. За неіснуючою (вигаданою) адресою. |

Продовження таблиці 1.1

| N | Назва ознаки | Зміст ознаки |
|---|---|---|
| | | 4. Наявність в установчих документах неправдивих даних про засновників і керівників клієнта. |
| 11. | "Штучне" створення статутного капіталу клієнта | "Штучне" створення статутного капіталу клієнта може відбуватися внаслідок проведених "циклічних" фінансових операцій із неодноразового поступового перерахування певної суми коштів з рахунку на рахунок кола юридичних осіб з призначенням платежу "поповнення статутного капіталу". |
| 12. | Керівник належить до соціально вразливих верств населення або є особою, яка зареєстрована на непідконтрольній Україні території | 1. Призначення на посаду керівника клієнта особи:<br>- яка належить до соціально вразливих верств населення (студенти, пенсіонери, тимчасово непрацюючі, знаходяться в декретній відпустці тощо);<br>- реально існуючого громадянина із специфічним соціальним статусом (малозабезпечені тощо);<br>- осіб молодого (до 20 років) чи похилого віку (після 75 років);<br>- яка зареєстрована на непідконтрольній Україні території (зона АТО, Крим). |
| 13. | Надання права управління рахунком іншим особам | 1. Надання права управління рахунком третім особам, які не пов'язані з клієнтом (не є посадовими особами, не отримують заробітної плати від нього), на підставі довіреностей. |
| 14. | Значний обсяг операцій з готівкою | 1. Проведення клієнтом фінансових операцій з готівкою на мільйонні суми, що не пов'язані з основним видом діяльності клієнта, та/або проведення значної кількості операцій з використанням карткових рахунків. |
| 15. | Зовнішньоекономічні розрахунки | 1. Новостворена компанія або незначний період діяльності клієнта.<br>2. Здійснення попередньої оплати за зовнішньоекономічними (імпортними) |

Продовження таблиці 1.1

| N | Назва ознаки | Зміст ознаки |
|---|---|---|
| | | контрактами при наявності інформації з відкритих джерел про невиконання нерезидентом своїх зобов'язань за іншими зовнішньоекономічними договорами.<br>3. Проведення розрахунків за зовнішньоекономічними контрактами (експортно-імпортними) на умовах передоплати, за якими не відбулася поставка товарів у визначені у таких договорах строки. |

Отже, згідно з таблиці1.1, у держави є список критеріїв для виявлення шахрайських операцій, але недостатньо інструментів.

## 1.2 Системи електронних платежів в Україні, як інструмент здійснення банківських операцій

Для характеристики досліджуваного механізму розглянемо поняття “електронний платіж”. Згідно з офіційними джерелами, електронний платіж – це оплата покупок за допомогою електронних засобів. Є багато систем, використання яких пов’язане з різними проблемами – в першу чергу, проблемами безпеки і збереження конфіденційності. Найбільш розповсюджені такі системи, як домашній банк, електронна сплата за квитки, оплата товарів в електронних магазинах і т.д. Електронні платежі являють собою необхідний елемент електронної комерції.

Для роботи електронних платежів необхідна спеціалізована система. В кожній країні світу розробкою та підтримкою системи електронних платежів

займається державний Національний банк. В Україні, як і будь-якій іншій країні існує свій Національний банк.

Відповідно до статті 11 Закону України "Про платіжні системи та переказ коштів в Україні" Національний банк України має право створювати системи міжбанківських розрахунків, системи роздрібних платежів та інші види платіжних систем. Національний банк України забезпечує безперервне, надійне та ефективне функціонування і розвиток створених ним платіжних систем. Створені Національним банком України платіжні системи є державними платіжними системами.

На сьогодні в Україні функціонують дві платіжні системи, платіжною організацією та розрахунковим банком яких є Національний банк України:

– Система електронних платежів (СЕП);

– Національна платіжна система "Український платіжний ПРОСТІР".

Згідно з темою дослідження нам треба детальніше розглянути саме СЕП. Тому почнемо саме з цієї системи.

Система електронних платежів Національного банку (СЕП) – це державна банківська платіжна система, що забезпечує проведення міжбанківського переказу через рахунки, відкриті в Національному банку України.

СЕП забезпечує здійснення розрахунків у межах України між банками як за дорученнями клієнтів банків, так і за зобов’язаннями банків та інших учасників системи. У СЕП виконуються міжбанківські перекази у файловому режимі та в режимі реального часу. Здійснення учасником початкових платежів у файловому режимі є обов’язковим, а в режимі реального часу – за його вибором. Крім того, учасник системи, який працює в СЕП у файловому режимі, має забезпечити приймання платежів, відправлених на його адресу іншими учасниками СЕП у режимі реального часу.

У файловому режимі обмін міжбанківськими електронними розрахунковими документами здійснюється шляхом приймання-передавання документів, згрупованих у файли. Тривалість технологічного циклу становить 15 – 20 хвилин. Кошти списуються з технічного рахунку учасника СЕП у момент приймання початкових платежів до Центру оброблення СЕП та зараховуються на технічний рахунок учасника-отримувача у момент надходження до Центру оброблення СЕП квитанції про успішне приймання файла платежів у відповідь.

У режимі реального часу кошти списуються з технічного рахунку учасника СЕП-платника і зараховуються на рахунок учасника-отримувача одночасно.

СЕП приймає початкові платежі від учасника системи в межах поточного значення його технічного рахунку. У СЕП немає пріоритетів оброблення платежів, крім черговості їх надходження.

СЕП визнана системно важливою платіжною системою в Україні. Системна важливість СЕП обумовлена тим, що вона забезпечує здійснення 97% міжбанківських переказів у національній валюті в межах України. СЕП є системою класу RTGS.

Система міжбанківських електронних платежів має трирівневу ієрархічну структуру.

На 1-му, верхньому рівні СЕП міститься Центральна розрахункова палата. Вона обслуговується програмно-технічним комплексом АРМ-1, що виконує такі основні функції:

– «пересилання» міжрегіональних електронних документів засобами електронної пошти Національного банку України;

– перевірку правильності формування електронних документів;

– формування й підтримання в робочому стані основних довідників НБУ;

– захист електронних документів і системи в цілому від несанкціонованого доступу;

– диспетчеризація (бухгалтерський технологічний контроль) проходження міжрегіональних платежів і синхронізація закриття дня банку.

На 2-му рівні мережі перебувають регіональні розрахункові палати, які обслуговуються своїми програмно-технічними комплексами АРМ-2.

АРМ-2 — це програмно-технічний комплекс — ПТК, установлений у РРП і призначений для обслуговування певної кількості банків цього регіону та організації взаємодії з іншими АРМ-2. РРП може експлуатувати один чи кілька АРМ-2 залежно від кількості банків регіону та активності проведення ними міжбанківських платежів.

Кожне АРМ-2 забезпечує виконання таких основних операцій:

– обмін електронними документами між самою РРП і банками — учасниками міжбанківських розрахунків;

– формування та відправлення міжрегіональних платежів до ЦРП;

– отримання міжрегіональних платежів від ЦРП та їх аналіз;

– обмін електронними документами з іншими АРМ-2 своєї РРП.

– захист електронних документів і їх обробка від несанкціонованого втручання.

На 3-му, нижньому рівні СЕП перебувають учасники міжбанківських електронних розрахунків, які діють на підставі угод із РРП на проведення розрахунків та Положення про міжбанківські розрахунки в Україні згідно з Регламентом функціонування мережі розрахункових палат України. Учасниками електронних платежів можуть бути, і здійснювати за допомогою СЕП міжбанківські розрахунки, будь-які кредитно-фінансові підприємства й організації, котрі мають відкриті КР у відповідних РУ НБУ та задовольняють вимоги, що їх висуває НБУ до учасників СЕП.

Юридичні особи ще не є учасниками мережі електронних платежів і можуть користуватися її послугами лише через посередництво безпосередніх «учасників», укладаючи з ними відповідні договори.

У розпорядження кожного з учасників платежів надається єдина копія програмно-технічного комплексу з умовною назвою АРМ-3(або інакше АРМ НБУ), через який банк обмінюється інформацією із СЕП за допомогою файлів, структура та функціональне призначення яких визначені в документі «Інтерфейс між САБ і АРМ-3 системи електронних платежів (СЕП)».

АРМ-3 на рівні банку — учасника розрахунків забезпечує виконання таких основних операцій:

– перевірку пакетів платіжних документів, які підготовлені панком, що експлуатує даний АРМ-3;

– обмін електронними документами з відповідною РРП;

– захист системи від несанкціонованого втручання.

Оскільки для передавання пакетів використовується система електронної пошти НБУ, то банк одночасно є абонентським вузлом цієї пошти, а АРМ-3 — одним із основних кінцевих користувачів цього вузла.

Відповідно АРМ-2 є кінцевим користувачем вузла 2-го рівня ЕП НБУ, а АРМ-1— один із основних користу-вачів Центрального вузла ЕП НБУ.

Між учасниками СЕП на різних рівнях циркулюють різного роду платіжні документи. Зокрема, на рівні КБ-РРП — електронні документи (ЕД) і документи на паперових носіях (ПД), а на Рівні РРП/ЦРП —ЕД.

Загальну структуру СЕП НБУ України можна подати у вигляді схеми, зображеній на рисунку 1.1.

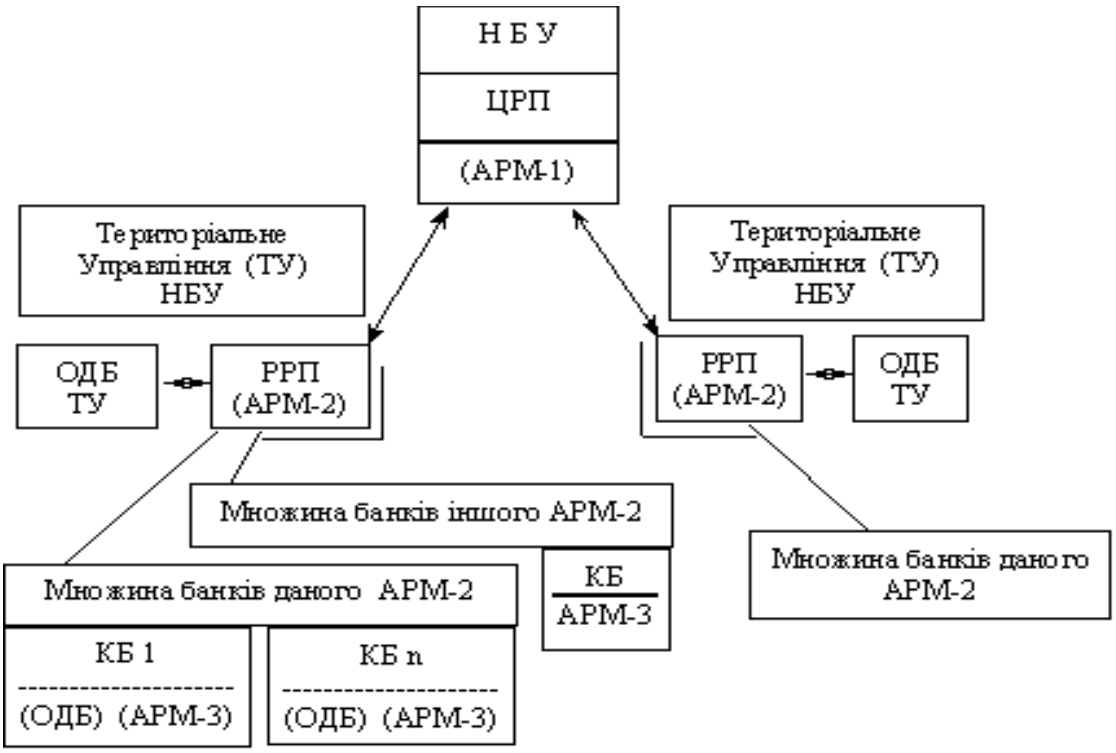



Рисунок 1.1 – Загальна структура СЕП України

Під час функціонування СЕП між її елементами циркулюють інформаційні потоки різного призначення. Одним із найважливіших і найпотужніших потоків інформації є потік, який складається з файлів платіжних документів, котрі ініціюють переміщення грошових коштів.

Файли-квитанції, які забезпечують і підтверджують правильність проходження потоків платіжних документів, у своїй сукупності створюють потоки підтвердження платежів і є другими після них за потужністю.

Така складна система, як СЕП потребує високого рівня синхронізації роботи її елементів, тому в системі існують потоки зазначеної синхронізації. Файли цих потоків несуть у собі повідомлення ЦРП (воно регламентує регіональним ланкам СЕП характер технологічного режиму та його зміну), а також дані про виб-рані режими роботи елементів системи.

У системі електронних міжбанківських платежів циркулюють також потоки аварійних сигналів і контрольної інформації, що присутні на всіх її рівнях.

Для шифрування і дешифрування ЕД застосовуються як програмні, так й апаратні методи.

Програмні методи - в АРМ-3 комерційних банків використовується програма шифрування і дешифрування з іменем TRESOR. Вона встановлюється для кожного КБ і його АРМ-3 індивідуально, захищена від можливості копіювання і працює (шифрує або розшифровує підготовлені чи отримані ЕД) з використанням спеціальних індивідуальних «ключів», серед яких немає двох однакових (їх через певний проміжок часу змінює служба безпеки НБУ).

Апаратні методи -система використання апаратного захисного обладнання — так званих модулів шифрування, які працюють з використанням шифрувального «ключа», що міститься на спеціальній пластиковій картці. Цю картку (вона має вбудований мікропроцесор і забезпечує високий рівень захи-сту «ключів») видають відповідальним працівникам банку, які мають право підпису банківських платіжних документів.

Використання самої картки передбачає введення додаткової системи паролів для авторизації її користувача.

Крім того, у СЕП існує також система захисту платіжних документів, яка ґрунтується на проведенні постійного оперативного банківського обліку, контролю та аналізу обсягів і напрямів руху і грошових коштів, які «несуть» ЕД на всіх етапах маршруту їх переміщення.

Проаналізуємо систему електронних платежів в Україні – “Простір”.

Український платіжний простір або «ПРОСТІР» (попередня назва «Національна система масових електронних платежів (НСМЕП)») — це внутрішньодержавна банківська багатоемітентна платіжна система, в якій розрахунки за товари та послуги, одержання готівки та інші операції

здійснюються за допомогою платіжних карток з магнітною смугою та EMV-чипом.

До системи “Простір” на даний момент підключені майже усі банки в Україні, їх детальний список зображено на рисунку 1.2.

| Банки, що підключені до Маршрутизатора ПРОСТІР | Обслуговують картки ПРОСТІР | | | | Можливість емісії карток | |
|---|---|---|---|---|---|---|
| | З магнітною смугою | | З чипом та магнітною смугою | | З магнітною смугою | З чипом та магнітною смугою |
| | ATM | POS | ATM | POS | | |
| ПАТ "Державний ощадний банк України" | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| АТ «Державний експортно-імпортний банк України" | ✓ | ✓ | ✓ | ✓ | ✓ | |
| ПАТ АБ "Укргазбанк" | ✓ | ✓ | ✓ | ✓ | ✓ | |
| ПАТ "СБЕРБАНК" | ✓ | ✓ | ✓ | ✓ | ✓ | |
| ПАТ КБ "ПРИВАТБАНК" | ✓ | ✓ | ✓ | ✓ | ✓ | |
| ПАТ АБ "Південний" | ✓ | ✓ | ✓ | ✓ | ✓ | |
| ПАТ "СКАЙ БАНК" | ✓ | | | | ✓ | |
| АТ "Райффайзен Банк Аваль" | ✓ | | ✓ | | ✓ | |
| Полікомбанк | ✓ | ✓ | | | ✓ | |
| ПАТ "Банк Восток" | ✓ | ✓ | ✓ | ✓ | ✓ | |
| ПАТ КБ "Глобус" | ✓ | ✓ | ✓ | ✓ | ✓ | |
| ПАТ КБ "ПРАВЕКС-БАНК" | ✓ | ✓ | ✓ | ✓ | ✓ | |
| ПАТ "БАНК СІЧ" | ✓ | ✓ | ✓ | ✓ | ✓ | |
| ПАТ "АЛЬПАРІ БАНК" | | | | | ✓ | |
| ПАТ "АЛЬФА-БАНК" | ✓ | ✓ | ✓ | ✓ | ✓ | |
| ПАТ КБ "Центр" | | | | | ✓ | |
| АТ "ОТП БАНК" | ✓ | | ✓ | | ✓ | |
| ПАТ "Таскомбанк" | ✓ | ✓ | ✓ | ✓ | | |
| ПАТ "РВС БАНК" | | | | | ✓ | ✓ |
| ПАТ "ЮНЕКС БАНК" | ✓ | ✓ | ✓ | ✓ | ✓ | |
| ПАТ "БАНК АЛЬЯНС" | | | | | ✓ | |
| ПАТ "ВЕРНУМ БАНК" | | | | | ✓ | |
| ПАТ "ДІВІ БАНК" | ✓ | ✓ | ✓ | ✓ | ✓ | |
| ПАТ "МЕГАБАНК" | | | | | ✓ | |

Рисунок 1.2 – Банки-учасники системи “Простір”

Метою створення системи “Простір” було розроблення та впровадження в Україні відносно дешевої надійно захищеної автоматизованої системи безготівкових розрахунків, яка в основному розрахована на роботу в режимі "офф-лайн".

Окрім того, технологією “Простір” передбачено також широке застосування платіжних карток юридичними особами — корпоративні та бізнес-картки.

Національний банк вважає створення “Простір” завершальною стадією у побудові системи електронних платежів в Україні. “Простір” за своєю масштабністю, складністю та обсягом витрат на створення, запровадження і експлуатацію значно перевищує існуючу систему електронних платежів НБУ. Враховуючи світовий досвід, в Україні має бути емітовано не менше 10 млн платіжних карток “Простір”, функціонувати щонайменше 100 тис. точок обслуговування (платіжних терміналів у торгівлі, на транспорті, в сфері послуг, банківських терміналів та банкоматів тощо).

У процесі вивчення теми потрібно засвоїти, що в системі “Простір” застосовують платіжні картки з вбудованими чіп-модуля-ми — смарт-картки. Смарт-картка, яка є носієм копії фінансової інформації в системі “Простір”, на відміну від пластикової картки з магнітною смугою, найбільш повно задовольняв вимоги безпеки. На її базі створена високоефективна офф-лайнова технологія, тобто така, що не потребує оперативного зв'язку з банківським рахунком під час виконання платіжних операцій (це дуже важливо, враховуючи недостатньо високу якість вітчизняних каналів зв'язку), яка значно зменшує експлуатаційні витрати. За рахунок цього участь у системі “Простір” може брати населення України з малими доходами (пенсіонери, студенти, школярі тощо), а швидкість обслуговування набагато більша, ніж при розрахунках готівкою та операціях з картками з магнітною смугою.

Усі картки, що застосовуються в системі “Простір”, можна умовно поділити на платіжні та службові. Студенти повинні вміти розрізняти такі платіжні інструменти, як електронний чек і електронний гаманець, знати визначення таких понять, як анонімний гаманець, персоналізований гаманець тощо.

Варто звернути увагу на те, що службові картки, залежно від призначення, поділяють на такі типи:

– модулі безпеки терміналів;

– трансферні (для інкасації інформації з терміналу в обслуговуючий банк чи процесинговий центр та передачі службової інформації від них у термінал);

– картки доступу.

Функціональні можливості кожного типу карток системи “Простір” визначають залежно від обраного чіпа. Максимальна кількість платіжних додатків (гаманців та чеків) на картці обмежена 30 додатками та залежить від обсягу енергонезалежної пам'яті чіпа.

Важливо знати, що організаційна структура системи “Простір” це сукупність визначених Платіжною організацією суб'єктів, їх функцій, прав і обов'язків, а також сукупність відносин, що виникають між ними під час проведення переказу коштів та забезпечення діяльності НСМЕП.

До складу НСМЕП належать:

– платіжна організація;

– члени платіжної системи;

– учасники платіжної системи.

Функції Платіжної організації системи “Простір” виконує Національний банк України. Членами НСМЕП можуть бути юридичні особи, які укладуть договір з Платіжною організацією про вступ до системи “Простір”. Член системи “Простір” може виконувати функції емітента та/або еквайра.

Учасниками системи “Простір” є юридичні або фізичні особи — суб'єкти відносин, що виникають при проведенні переказу коштів, ініційованого за допомогою платіжного додатка картки системи “Простір” (розрахунковий банк, головний процесинговий центр, регіональні процесингові центри, процесингові центри банківського рівня, держателі платіжних карток, підприємства торгівлі та послуг тощо).

Моделі роботи членів системи “Простір” такі:

– емітент;

– еквайр;

– емітент та еквайр.

СЕП – дуже потужна та завантажена система.

Згідно з даними сайту НБУ, станом на 01 жовтня 2017 року учасниками СЕП були 187 установ, із них:

– 93 – банки України;

– 66 – філії банків України;

– 27 – Державна казначейська служба України та її органи;

– 1 – Національний банк України.

За 9 місяців 2017 року СЕП було оброблено 242 395 тис. початкових платежів на суму 14 908 млрд. грн., у тому числі:

– у файловому режимі – 241 704 тис. початкових платежів на суму 10 026 млрд. грн.;

– у режимі реального часу – 691 тис. початкових платежів на суму 4 882 млрд. грн.

Переважна більшість початкових платежів надіслана до СЕП банками України та їх філіями – 226 952 тис. початкових платежів (94% від їх загальної кількості).

За 9 місяців 2017 року в СЕП у середньому за день оброблялося 1 310 тис. початкових платежів на суму 80 млрд. грн.

Середньоденний залишок коштів на рахунках учасників СЕП становив 86,64 млрд. грн., а середньодобовий коефіцієнт обігу коштів за рахунками учасників системи – 0,93.

Для аналізу СЕП розглянемо функції кожного рівня архітектур даної системи.

Найвищий рівень в архітектури СЕП це АРМ-1, потім АРМ-2 та АРМ-3.

Функції АРМ-1:

– Синхронізація роботи СЕП;
– Нагляд за функціонуванням РРП;
– Складання балансу міжрегіональних платежів;
– Надання інформації про функціонування СЕП в цілому;
– Надання інформаційних довідок;
– Захист інформації.

Функції АРМ-2:

– Передача платіжних документів та іншої інформації, що обробляється СЕП, комерційним банкам – учасникам розрахунків та іншим РРП;
– Передача інформації, що не є платіжними документами, але обробляється в СЕП (АРМ-1);
– Виконання бухгалтерського та технологічного контролю за проходженням платежів;
– Ведення технічних коррахунків банків учасників СЕП;
– Надання звітності за підсумками проходження платежів;
– Підготовка інформації для відображення на реальних коррахунках учасників СЕП;
– Надання довідникової інформації.

Функції АРМ-3:

– Прийняття платіжних документів та іншої інформації, що обробляється СЕП, від РРП;

– Передача платіжних документів та іншої інформації, що обробляється СЕП в РРП для обробки в АРМ-2;

– Відкриття та закриття операційного дня;

– Архівація інформації за операційний день;

– Захист інформації, що передається засобами криптографії та накладання цифрового підпису;

– Сервіс: друк виписки про стан технічного коррахунку тощо.

Отже, згідно даних проаналізованих систем, кожна з них має потужну та складну технічну базу, але в них не передбачені ризики здійснень шахрайських операцій. Опираючись на це, вирішено розробити прототип захищеної від шахрайських операцій платіжну систему.

## 1.3 Постановка завдання для розробки автоматизованого рішення виявлення шахрайських операцій

Завданням автоматизації є розробка рішення для виявлення шахрайських операцій клієнтів банку. Для цього необхідно розробити веб-додаток, який складається з серверної та клієнтської частин.

Серверна частина системи буде розроблена з використанням мови програмування Ruby, MVC-фреймворку Ruby on Rails, бази даних PostgreSQL та середовищем для розробки RubyMine.

Клієнтська частина - за допомогою мов HTML, CSS та Javascript. Бібліотеками для розробки обрано React, Redux та React Bootstrap.

Згідно з вимогами розробляємого додатку, він повинен виконувати наступні функції:

– надавати можливість регістрації користувачу

– надавати можливість авторизуватися та аутенфікуватися в системі;

– редагувати дані власного профіля (ім’я, фамілію, дату народження)

– добавляти, редагувати та видаляти картки;

– проводити електроні платежі (транзакції);

– дивитись історію своїх транзацій, як вхідних так і здійсненних користувачем безпосередньо;

– подавати скарги на інших користувачів після проведення транзакції, у разі виникнення підозри про шахрайство.

Грунтуючись на основні критерії Департаменту фінансового моніторингу України щодо шахрайських операцій та інформаційну базу, що ми можемо використовувати для розробки алгоритму було вирішено використовувати наступні дані для виявлення підробних операцій:

– скарги клієнтів про проведені транзакції;

– місцеположення користувача під час проведення операції;

– ip-адресу користувача.

На заснуванні цих даних можна впізнати транзакцію як шахрайську по наступному сценарію:

– якщо на користувача вже було подано більш ніж 3 скарги і він знаходиться в чорному списку системи;

– якщо місцеположення користувача та його ip-адреса постійно кардинально змінюються (країна, місто), то це можна вважати спробою замаскувати транзакцію, що ускладнить роботу правоохоронних структур у пошуку зловмисника.

Дана система призначена для клієнтів банку, які активно користуються системами електронних платежів. Інтегрування даного додатку с платіжною

системою банку захистить її користувачів від махінацій за сторони шахраїв, що збережи їхні кошти та репутацію банку.

# РОЗДІЛ 2 ПРОЕКТУВАННЯ ПРОТОТИПУ ДОДАТКУ АВТОМАТИЗАЦІЇ ВИЯВЛЕННЯ ШАХРАЙСЬКИХ ОПЕРАЦІЙ

## 2.1 Вибір технологій для розробки додатку

У процесі створення й реалізації автоматизованих інформаційних технологій управління дуже важливо правильно обрати архітектуру інформаційної системи. Архітектура інформаційної комп'ютерної системи будується на основі апаратної частини (ЕОМ), телекомунікаційного і програмного забезпечення. Рівень розвитку кожної із складових визначений досконалістю інформаційної системи, технологією обробки даних, що зумовило виникнення таких схем обробки даних: файл-сервер, клієнт-сервер, Internet-система; сховище даних і система оперативної аналітичної обробки даних.

Проаналізувавши всі вище зазначені вимоги, які ставляться до нашої автоматизованої системи стає зрозуміло, що доцільно використати клієнт-серверну архітектуру.

Архітектура «клієнт-сервер» – це мережева архітектура, яка може застосовуватися як до фізичних приладів, так і до програмного забезпечення в залежності від того, як розподілені логічні компоненти додатку між клієнтом та сервером. При цьому сама архітектура складається з наступних елементів: сервер бази даних – прилад, який здійснює управління зберіганням, доступом, захистом, а також резервним копіюванням даних; сервер додатків – прилад, який виконує визначення бізнес-правила; клієнт – прилад, який представляє собою користувацький інтерфейс; мережа і комунікаційне програмне забезпечення – це обладнання, канали передачі даних і програмного забезпечення, який здійснює передачу запитів і відповідей між клієнтами і

сервером мережевих протоколів. Розглянувши загальні положення архітектури «клієнт-сервер» ми можемо подувати та наочно представити загальну схему нашої системи:

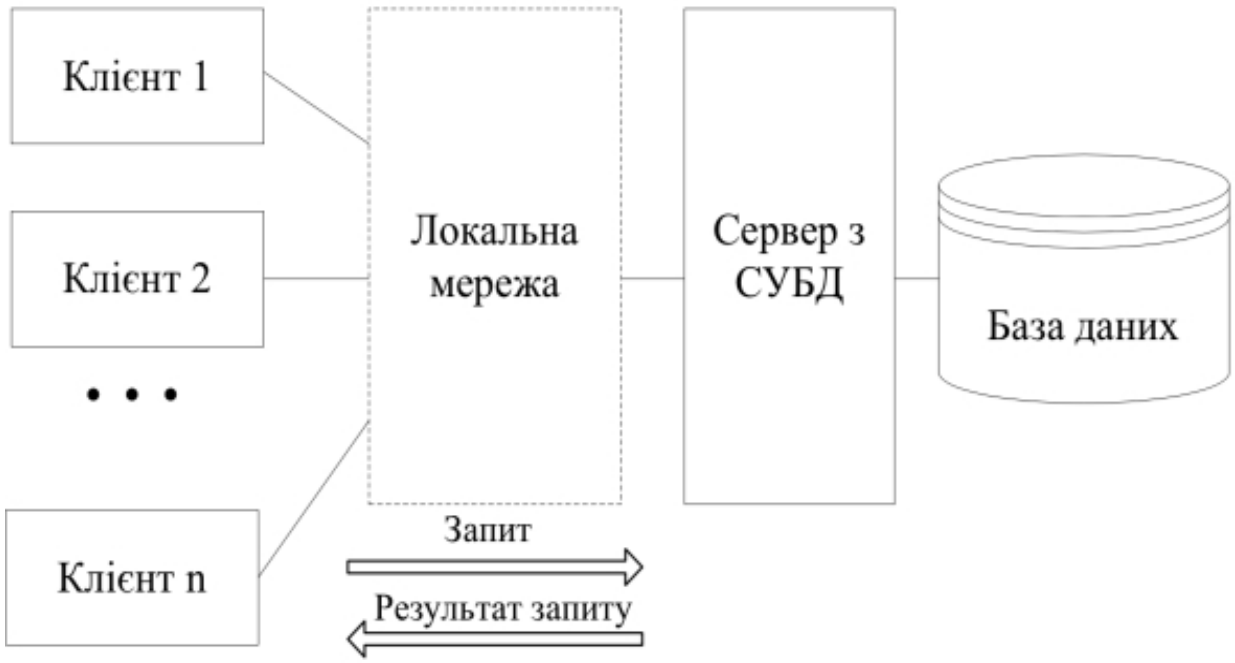



Рис. 2.1 - Загальна схема побудови системи з архітектурою “Клієнт-сервер”

Таблиця 2.1 – Переваги побудованої системи на основі архітектури “клієнт-сервер”

| № | Перевага | Значення |
|---|---|---|
| 1. | Незалежність мережі | Для роботи системи необхідне лише Інтернет-з’єднання |
| 2. | Максимізація потужності системи | Система має в розпорядженні потужності, які дозволяють виконувати роботу без монополізації ресурсів, так як у клієнтська частина додатку використовую ресурси пристрою користувача. |

Продовження таблиці 2.1

| | | |
|---|---|---|
| 3. | Відкритість системи | При побудові нашої інформаційної системи за рахунок можливості інтегрованості та взаємодії передбаченої в архітектурі «клієнт-сервер» ми можемо використовувати апаратне та програмне забезпечення різних виробників. |
| 4. | Легкість нарощування системи | Представлену систему не важко модернізувати, як тільки змінюються певні вимоги до неї. |
| 5. | Індивідуальне робоче середовище клієнта | Клієнтська частина системи є унікальною для кожного користувача. |

Для розробки серверної частини додатку ми вирішили використовувати мову програмування Ruby та фреймворк Ruby on Rails. Ruby - це швидка динамічна мова з відкритим вихідним кодом з упором на простоту і продуктивність. Він має елегантний синтаксисом, який приємно читати і легко писати. Дана мова при виконанні інтерпретується в мову С, яка в свою чергу компілюється в байт-код, яка для компьютер є “рідною” мовою. Для порівняння - інша популярна мова C# - відразу компілюється в байт-код перед запуском додатку.

Ruby on Rails - фреймворк, написаний на мові програмування Ruby, який реалізує архітектурний шаблон MVC(Model-View-Controller) для веб-додатків, а також забезпечує їх інтеграцію з веб-сервером і сервером баз даних. Є відкритим програмним забезпеченням.

Базу даних для нашого додатку ми обрали PostgreSQL - об'єктно-реляційну систему управління базами даних, так як це найшвидша на даний момент база даних, та також має відкритий вихідний код.  Тим паче ця БД має реалізації для всіх UNIX-подібних систем, включаючи AIX, різні BSD-системи, HP-UX, IRIX, Linux, macOS, Solaris / OpenSolaris, Tru64, QNX, а також для Microsoft Windows.

Для зв’язку нашого додатку з базою даних ми вирішили використовувати ORM-систему Active Record.

ORM (англ. Object-Relational Mapping, укр. Об'єктно-реляційне відображення, або перетворення) - технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних». Існують як пропрієтарні, так і вільні реалізації цієї технології.

Схема Active Record - це підхід до доступу до даних в базі даних. Таблиця бази даних або подання обгорнуті в класи. Таким чином, об'єктний екземпляр прив'язаний до єдиному рядку в таблиці. Після створення об'єкта новий рядок буде додаватися до таблиці на збереження. Будь завантажений об'єкт отримує свою інформацію від бази даних. Коли об'єкт оновлений, відповідний рядок в таблиці також буде оновлено. Клас обгортки реалізує методи засоби доступу або властивості для кожного стовпця в таблиці або поданні.

Клієнтська частина додатку являє собою веб-додаток(сайт) та буде розроблена як SPA(Single Page Application). Для реалізації цього ми будемо використовувати HTML, CSS та JavaScript. А також бібліотеку ReactJS. Дана технологія найсучасніша та найпопулярніша на даний момент та дозволяє швидко розробляти клієнтський інтерфейс додатку.

## 2.2 Проектування структури БД та архітектури додатку

Для розробки нашого додатку треба спочатку визначитись з архітектурою системи. Для цього спочатку треба спроектувати базу даних. Для реалізації нашої БД, ми будемо використовувати PostgreSQL версії 9.5.

Згідно з нашими цілями база даних цієї системи повинна мати наступні таблиці: User, Card, Transaction, Account, Claim, Blacklist та додаткові зв’язуючі таблиці: TransactionAccount.

Далі про кожну таблицю більш детально.

User - дана таблиця буде містити в собі дані про користувача. Поля цієї таблиці представлені в таблиці 2.2.

Таблиця 2.2 - Поля таблиці User.

| Назва поля | Тип даних | Опис |
|---|---|---|
| id | Integer | Унікальний ідентифікатор запису в таблиці |
| first_name | String | Ім’я користувача |
| last_name | String | Прізвище користувача |
| birth_date | Date | Дата народження користувача( банківськими картками можуть користуватися лише особи, які досягли 16 років) |
| email | String | Електрона адреса користувача для авторизації в додатку |
| card_ids | Array | Масив унікальних ідентифікаторів карток користувача |

Card - дана таблиця буде містити в собі дані про картку користувача. Так як ми розробляємо лише прототип додатку, то будемо вважати що кожна карта має окремий рахунок. Поля цієї таблиці представлені в таблиці 2.3.

Таблиця 2.3 - Поля таблиці Card.

| Назва поля | Тип даних | Опис |
|---|---|---|
| id | Integer | Унікальний ідентифікатор запису в таблиці |
| user_id | Integer | Ідентифікатор користувача картки |
| number | Integer | 16-значний номер картки, який має бути валідним за алгоритмом Луна |
| cvv2 | Integer | 3-значний код для перевірки справжності картки |
| expiry_date | Date | Дата, до якої картка є дійсною |
| account_id | Integer | Ідентифікатор рахунку, з яким зв’язана дана картка |

Transaction - дана таблиця буде містити в собі дані про всі транзакції, що проводять користувачі. Поля цієї таблиці представлені в таблиці 2.4.

Таблиця 2.4 - Поля таблиці Transaction.

| Назва поля | Тип даних | Опис |
|---|---|---|
| id | Integer | Унікальний ідентифікатор в таблиці |
| account_from_id | Integer | Ідентифікатор рахунку користувача, який переводить кошти |
| account_to_id | Integer | Ідентифікатор рахунку користувача, якому переводять кошти |
| amount | Integer | Сума коштів, яку переводять в даній транзакції, має бути додатнім числом(>0) |

Продовження таблиці 2.4

| Назва поля | Тип даних | Опис |
|---|---|---|
| result | Enum | Результат транзакції(відхилино, схвалено) |
| loc | String | Географічні координати користувача, при створенні транзакції |
| city | String | Місто користувача, при створенні транзакції |
| country | String | Країна користувача, при створенні транзакції |
| ip | String | ip-адреса користувача, при створенні транзакції |

Account - дана таблиця буде містити в собі дані про рахунок користувача. Поля цієї таблиці представлені в таблиці 2.5.

Таблиця 2.5 - Поля таблиці Account.

| Назва поля | Тип даних | Опис |
|---|---|---|
| id | Integer | Унікальний ідентифікатор в таблиці |
| card_id | Integer | Ідентифікатор картки користувача, яка зв’язана з даним рахунком |
| amount | Integer | Сума коштів на рахунку |
| transaction_ids | Array | Масив ідентифікаторів транзакцій, які зв’язані з даним рахунком |

Claim - дана таблиця буде містити в собі дані про всі скарги на користувача. Поля цієї таблиці представлені в таблиці 2.6.

Таблиця 2.6 - Поля таблиці Claim.

| Назва поля | Тип даних | Опис |
|---|---|---|
| id | Integer | Унікальний ідентифікатор в таблиці |
| user_id | Integer | Ідентифікатор користувача, на якого надійшла скарга |
| description | Text | Опис скарги у вигляді текстового повідомлення |

Blacklist - дана таблиця буде містити в собі користувачів, які вважаються шахраями. До даної таблиці вони будуть потрапляти, якщо на них надійшла велика кількість скарг. Її можна вважати зв’язуючою таблицею між користувачем та скаргами. Поля цієї таблиці представлені в таблиці 2.7.

Таблиця 2.7 - Поля таблиці Blacklist.

| Назва поля | Тип даних | Опис |
|---|---|---|
| id | Integer | Унікальний ідентифікатор в таблиці |
| user_id | Integer | Ідентифікатор користувача, який вважається шахраєм |
| claim_ids | Array | Масив ідентифікаторів скарг |

TransactionAccount - це зв’язуюча таблиця між Transaction та Account, так як не потрібен прямий зв’язок між цими таблицями. Поля цієї таблиці представлені в таблиці 2.8.

Таблиця 2.8 - Поля таблиці TransactionAccount.

| Назва поля | Тип даних | Опис |
|---|---|---|
| id | Integer | Унікальний ідентифікатор в таблиці |
| transaction_id | Integer | Ідентифікатор транзакції, пов’язаної з рахунком |
| account_id | Integer | Ідентифікатор рахунку користувача |

Згідно з описаними вище таблицями, ми можемо розробити детальну схему нашої бази даних.

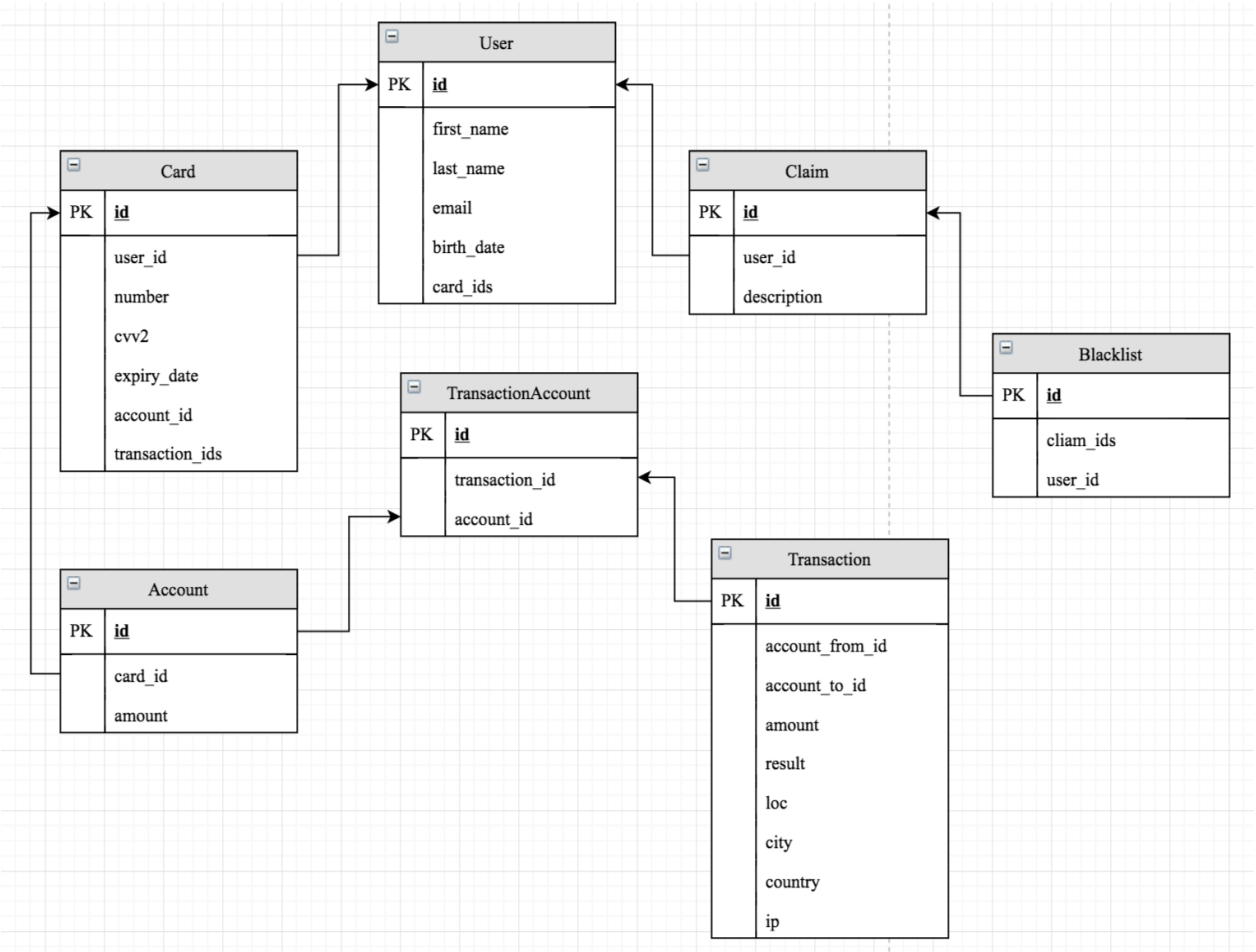



Рис 2.2 - Схема бази даних системи

Згідно зі схемою бази даних у системи доволі прозора та зрозуміла архітектура. Це дає змогу розробити прототип інтерфейсу додатка.

## 2.3 Проектування інтерфейсу додатку

Щоб розробити інтерфейс клієнтської частини додатку, треба визначити які саме сторінки повинен мати наш веб-додаток. Для цього використаємо популярний спосіб опису вимог до системи User Story.

Призначені для користувача історії (англ. User Story) - спосіб опису вимог до розроблюваної системі, сформульованих як одне або більше пропозицій на повсякденному або діловою мовою користувача. Призначені для користувача історії використовуються гнучкими методологіями розробки програмного забезпечення для специфікації вимог. Кожна призначена для користувача історія обмежена в розмірі та складності. Часто історія пишеться на маленькій паперовій картці. Це гарантує, що вона не стане занадто великий. В екстремальних програмуванні призначені для користувача історії пишуться користувачами (замовниками) системи. У методології SCRUM - пишуться або схвалюються роллю власника продукту. Для замовників призначені для користувача історії є основним інструментом впливу на розробку програмного забезпечення.

Призначені для користувача історії - швидкий спосіб документувати вимоги клієнта, без необхідності розробляти великі формалізовані документи і згодом витрачати ресурси на їх підтримку. Мета користувальницьких історій полягає в тому, щоб бути в змозі оперативно і без накладних витрат реагувати на швидко змінюються вимоги реального світу.

Отже, згідно з нашими вимогами до додатку, маємо наступні користувацькі історії:

– користувач може зареєструватись в системі;

– користувач може авторизуватись в системі;

– користувач може переглянути власний кабінет;

– користувач може свої дані(ім’я, прізвище, дата народження)

– користувач може додавати, змінювати та видаляти картки;

– користувач може продивлятись історію транзакцій для кожної картки;

– користувач може проводити транзакції;

– користувач може подати скаргу на іншого користувача після здійснення транзакції.

Для кожної користувацької історії повинен існувати функціонал. Згідно перерахованих вище історій побудуємо прототипи сторінок.

Щоб користатися додатком, користувач повинен авторизуватись в системі, якщо в нього є обліковий запис. Якщо його немає - треба зареєструватись. На рисунку 2.3 зображено прототип сторінки реєстрації.

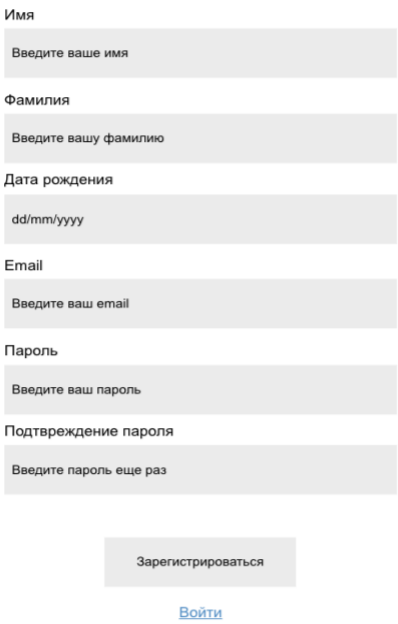



Рисунок 2.3 - Прототип сторінки реєстрації

Згідно з прототипом користувач має заповнити наступні поля форми:

– Ім’я;
– Прізвище;
– Дату народження;
– Електрону пошту;
– Пароль;
– Підтвердження паролю.

Якщо в користувача вже створено обліковий запис - він може просто увійти в систему за допомогою електронної пошти та паролю. Сторінка авторизації зображена на рисунку 2.4.

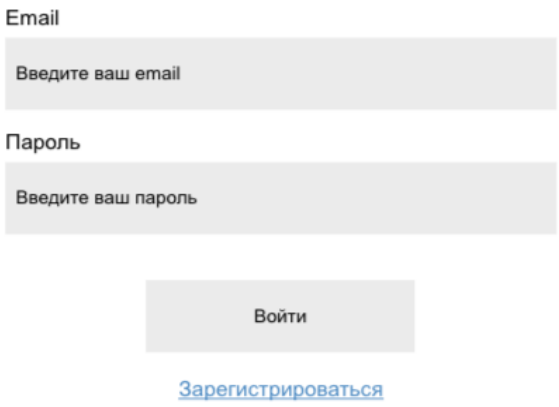



Рисунок 2.4 - Прототип сторінки авторизації

Після виконання реєстрації користувач потрапить на головну сторінку. Вона зображена на рисунку 2.5. На даному прототипі зображено надпис привітання та кнопка “Додати картку”, яка виводить модальне вікно з формою картки(рисунок 2.6).

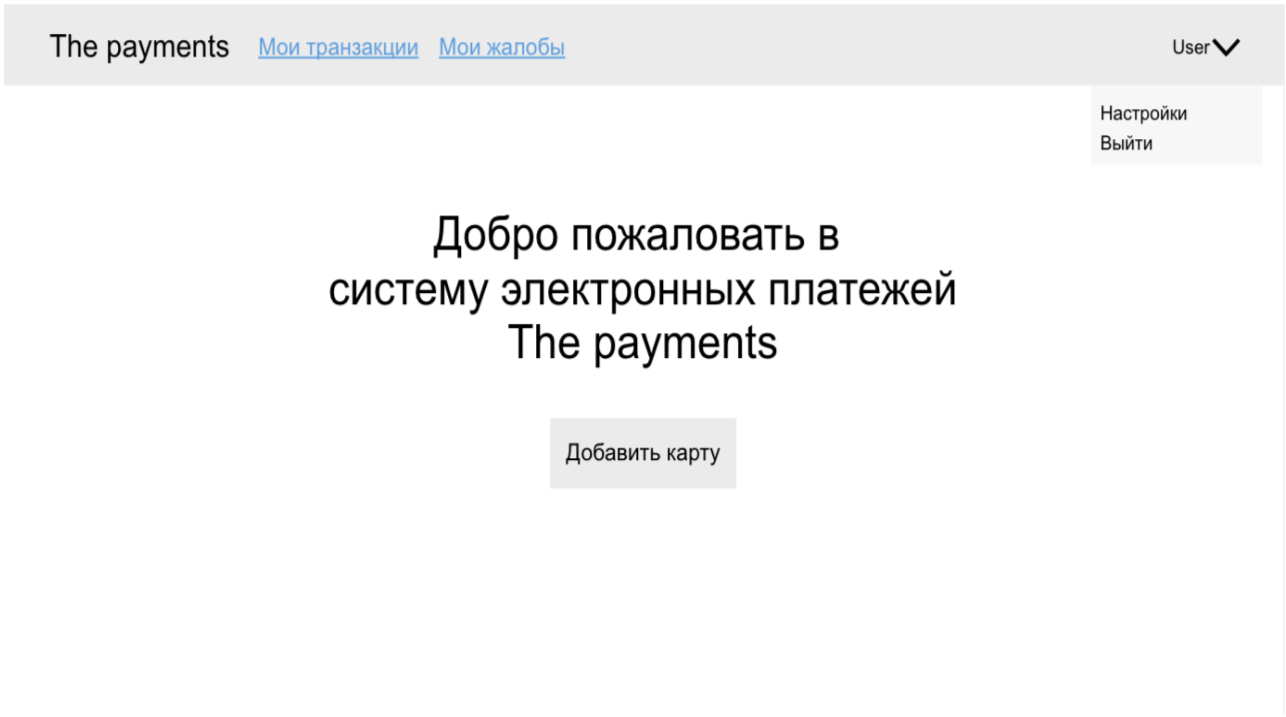



Рисунок 2.5 - Прототип головної сторінки

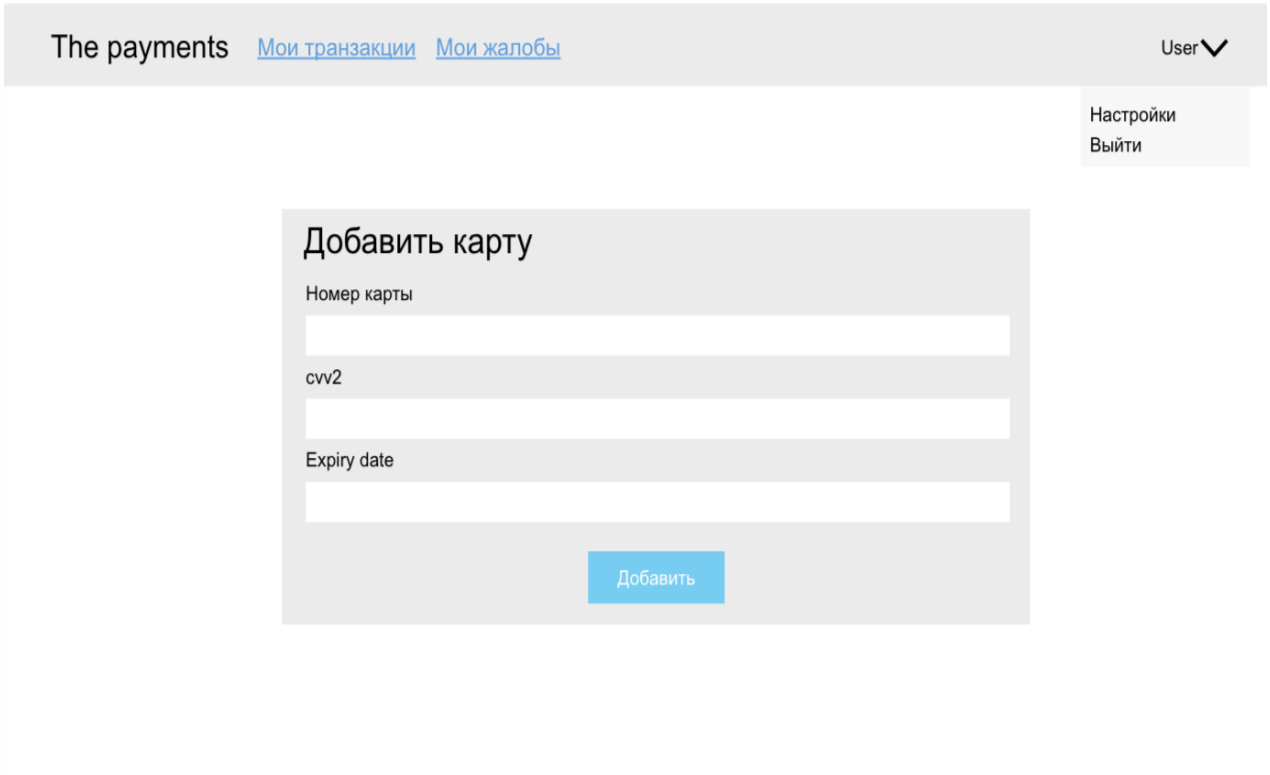



Рисунок 2.6 - Прототип модального вікна з формою додавання картки

Якщо в користувача вже є картки та була створена хоча б одна транзакція, він побачить сторінку, прототип якої зображено на рисунку 2.7, на якій виводиться список всіх карток користувача та кнопка “Створити транзакцію”, що активує модальне вікно з потрібною формою. Ця дія зображена на рисунку 2.8. В цій формі користувачу потрібно обрату одну зі своїх форм, за допомогою селекту, ввести карту отримувача та необхідну суму. Якщо отримувач знаходиться в чорному списку, появиться попередження, що він шахрай.

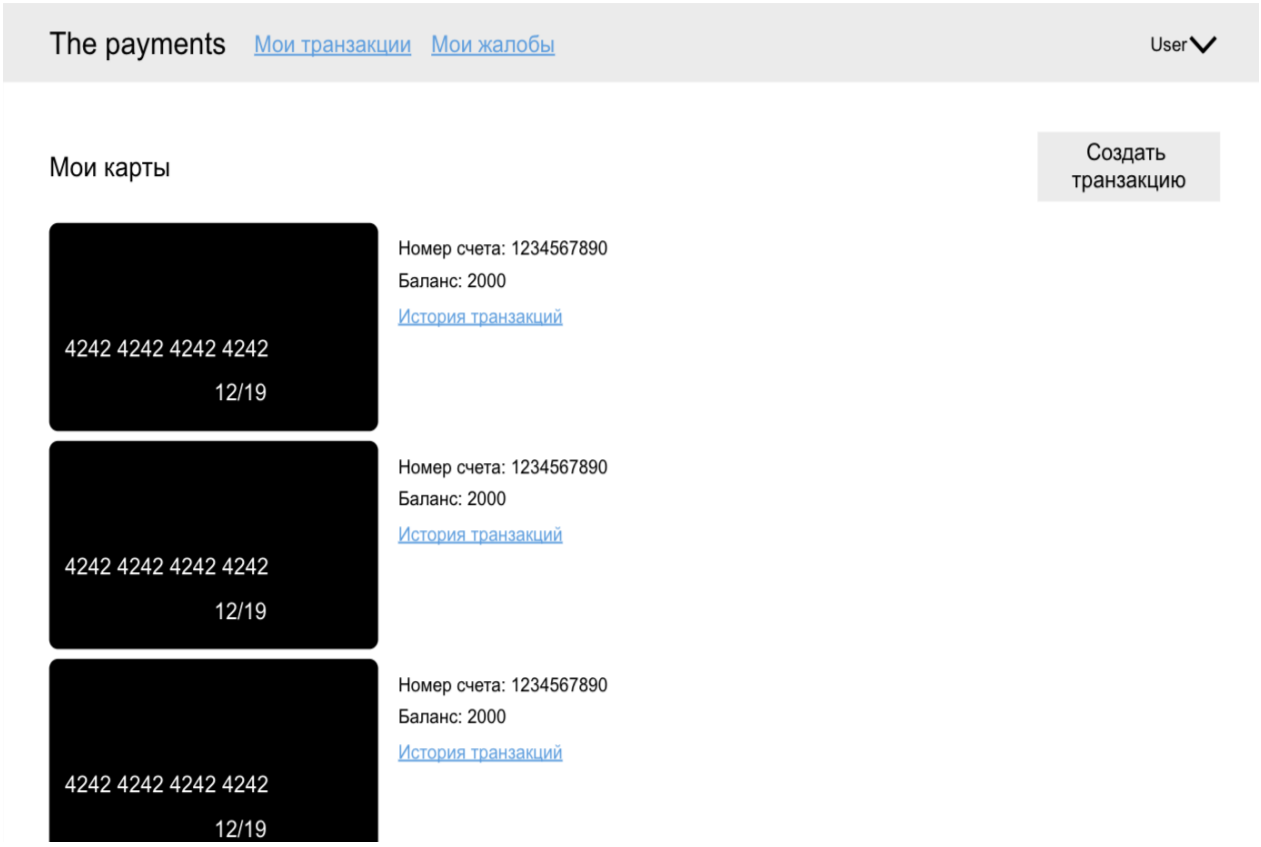



Рисунок 2.7 - Прототип головної сторінки

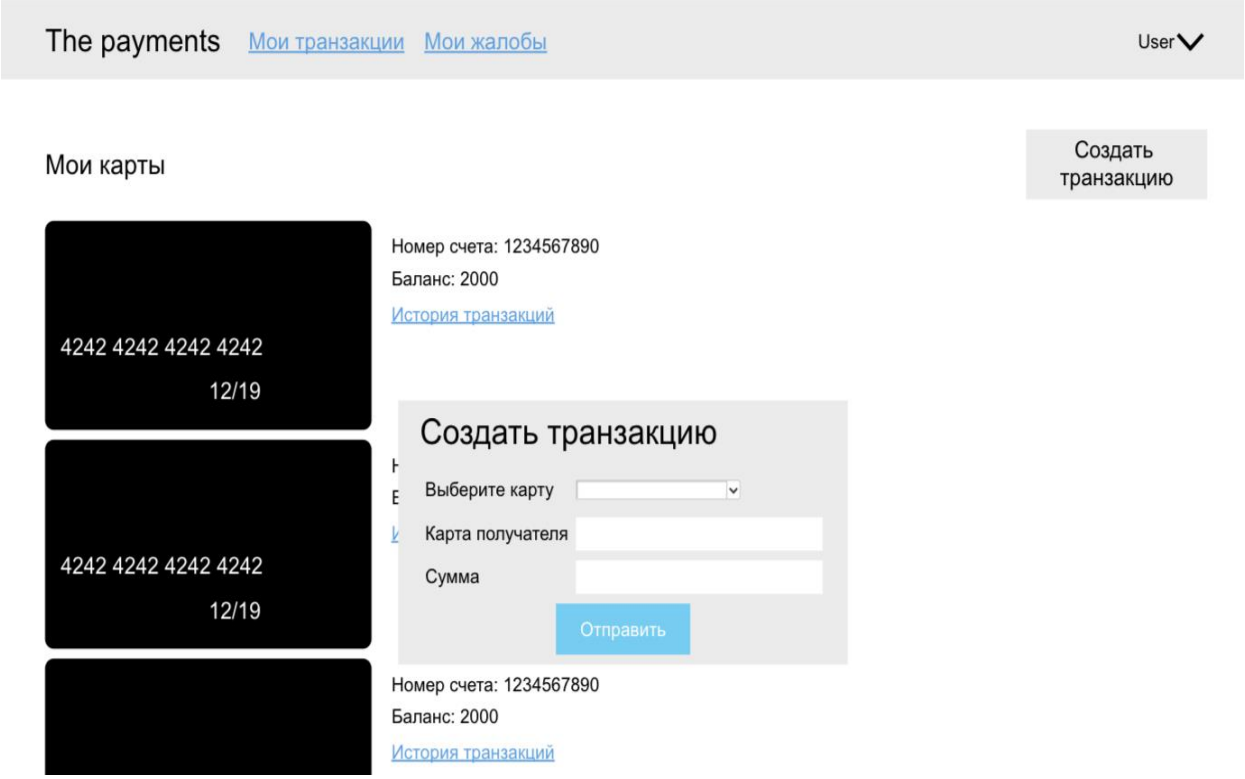



Рисунок 2.8 - Прототип модального вікна з формою створення транзакції

Згідно з прототипами в шапці сайту знаходяться логотип, посилання на сторінки транзакцій, скарг та меню користувача з двома підпунктами: “налаштування” та “вихід”. При кліку на “налаштування”, користувач перейде на сторінку, зображену на рисунку 2.9. Дана сторінка складається з двох: список карток та профіль. По замовчуванням користувач потрапить на представлення списку карток, в якому він може додати картку, переглянути існуючі, відкрити список транзакцій по кожній (рисунок 2.10), чи видалити будь-яку з них. Також на цій сторінці є кнопка додавання картки, яка активує модальне вікно, що зображено на рисунку 2.6.

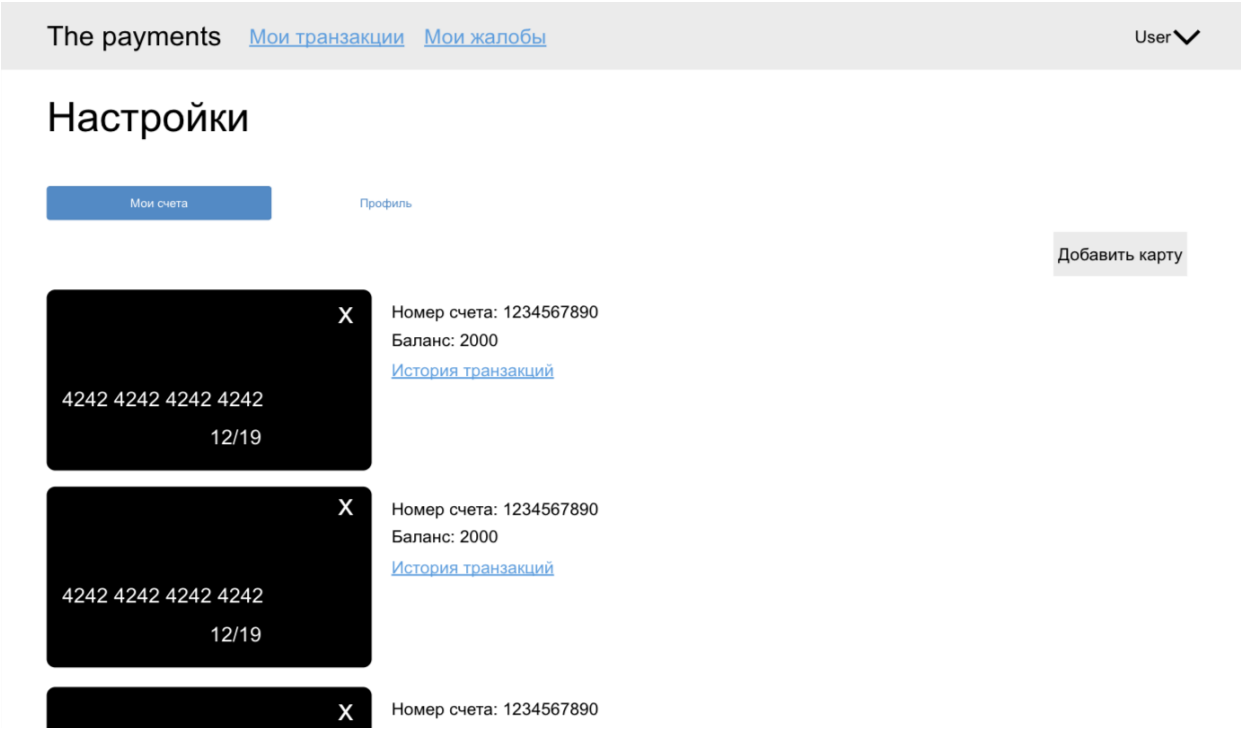



Рисунок 2.9 - Прототип сторінки налаштувань

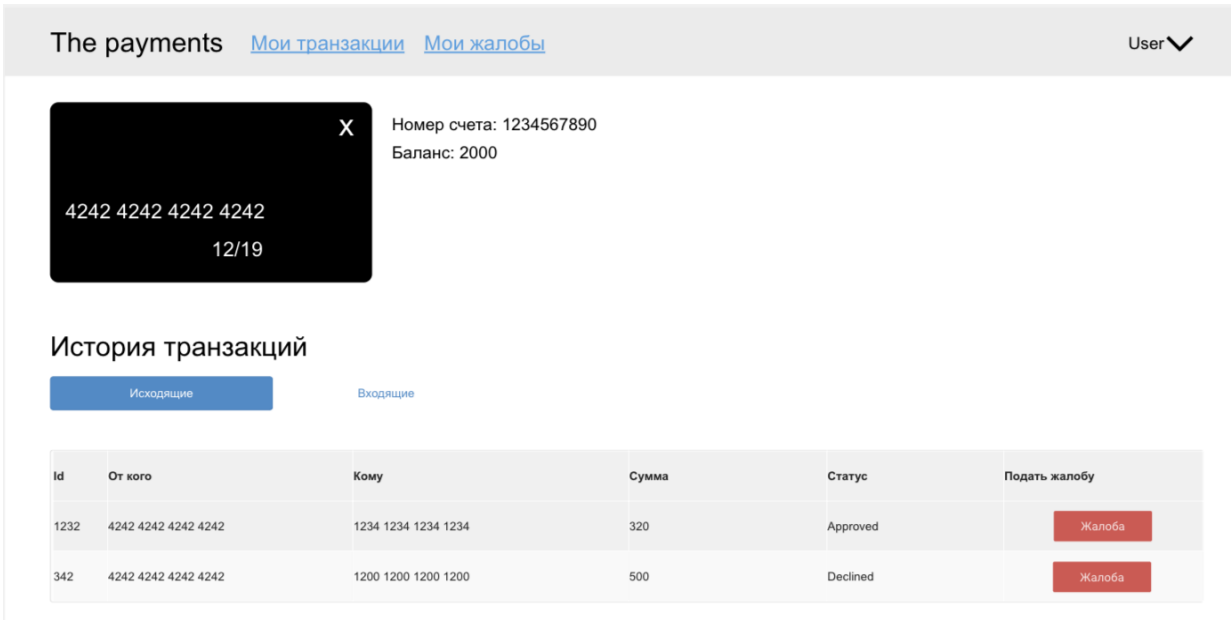



Рисунок 2.10 - Прототип сторінки історії транзакцій по картці

Згідно попереднього рисунку, користувач може проглянути відправлені/отримані транзакції та подати скаргу по кожній з них. Остання дія зображена на рисунку 2.11.

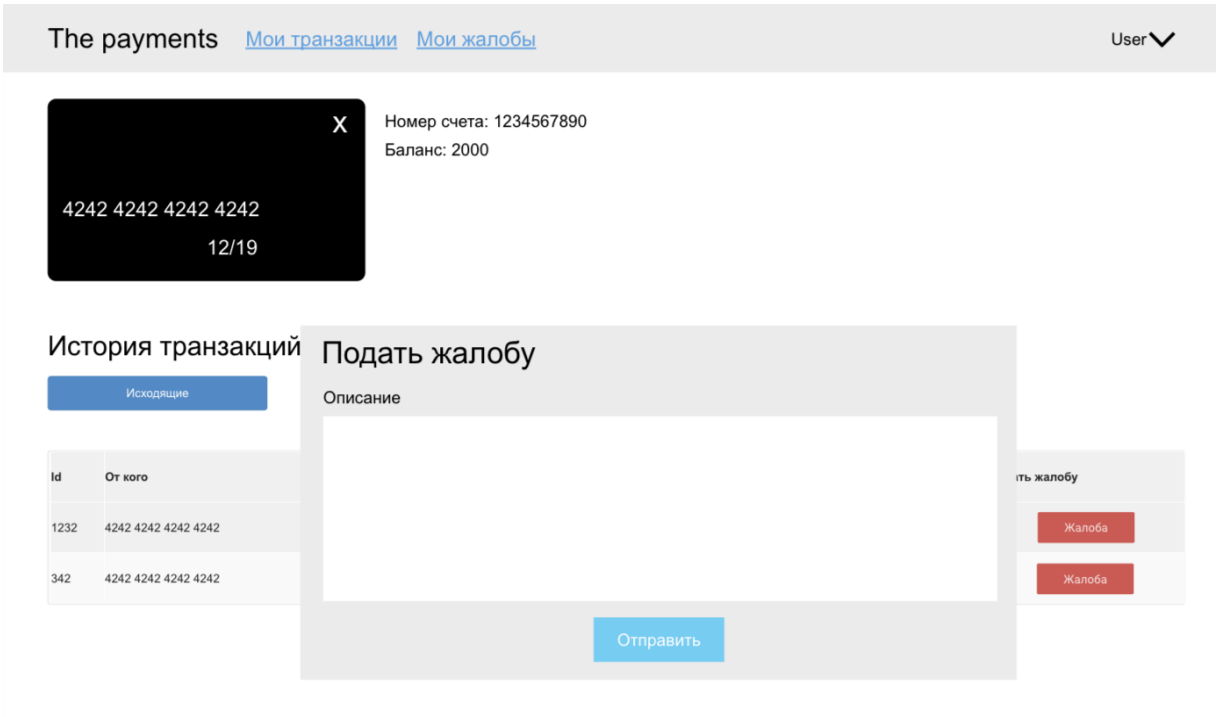



Рисунок 2.11 - Прототип модального вікна створення скарги

Користувач може переглянути список поданих ним скарг. Для цього в системі існує окрема сторінка, посилання на яку є в навігації сайту. Прототип даної сторінки представлено в рисунку 2.12.

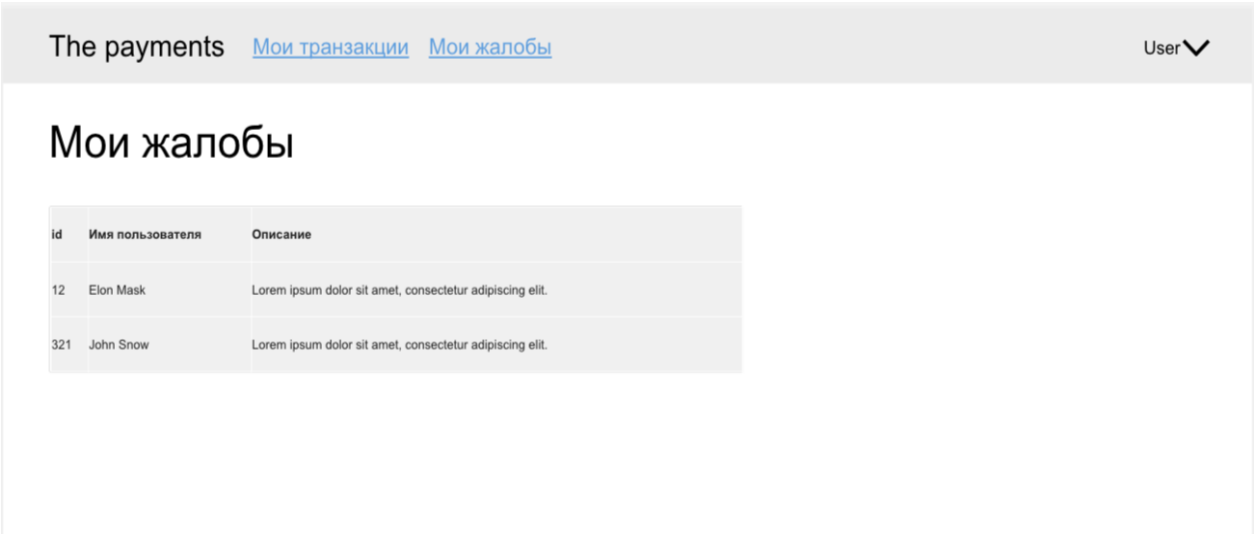



Рисунок 2.12 - Прототип сторінки скарг

Згідно з останнім пунктом навігації, користувач може потрапити на сторінку всіх створених чи отриманих ним транзакцій. Прототип цієї сторінки зображено на рисунку 2.13.

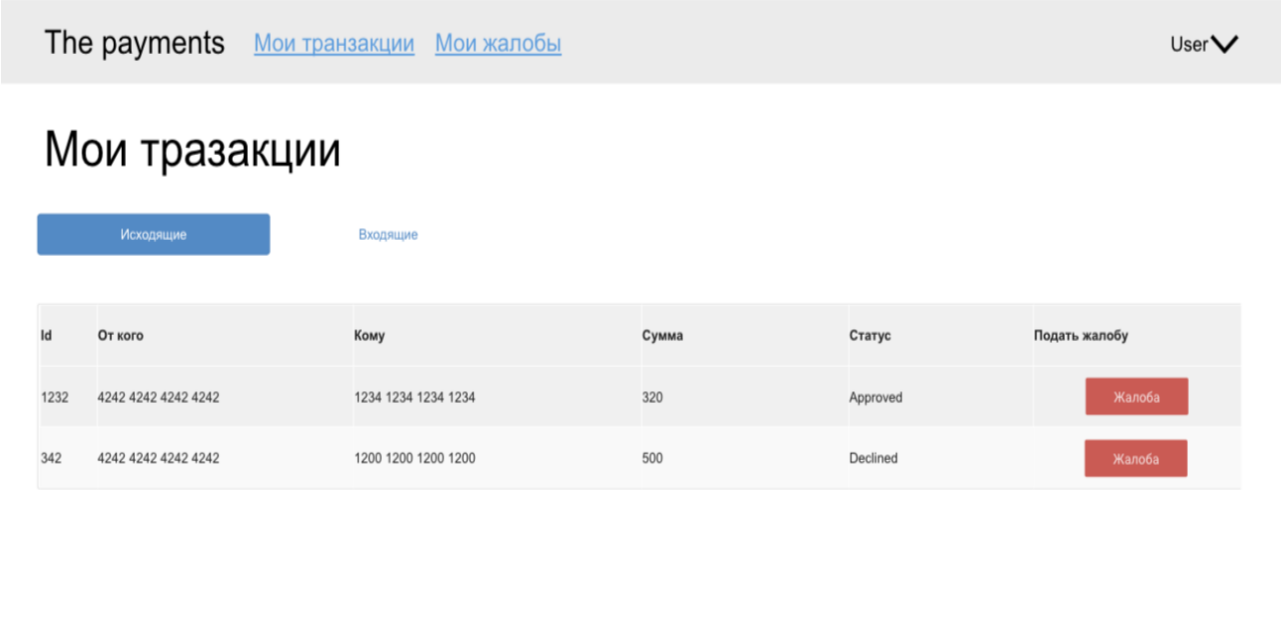



Рисунок 2.13 - Прототип сторінки тракзакцій

# РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОТОТИПУ ДОДАТКУ АВТОМАТИЗАЦІЇ ВИЯВЛЕННЯ ШАХРАЙСЬКИХ ОПЕРАЦІЙ

## 3.1 Розробка серверної частини додатку

Для розробки серверної частини нашого додатку було обрано мову програмування Ruby та фреймворк Ruby on Rails. Це дозволяє нам розробляти нашу систему достатньо швидко та зручно.

Як середовище розробки ми обрали RubyMine. RubyMine - комерційна IDE для розробки програмного забезпечення на Ruby компанії JetBrains. RubyMine створений на основі IntelliJ IDEA того ж виробника. Підтримує популярні бібліотеки, які використовуються в Ruby-додатках (в тому числі Bundler, RSpec, Shoulda, Cucumber, Git). Приклад вікна цієї програми відображено на рисунку 3.1.

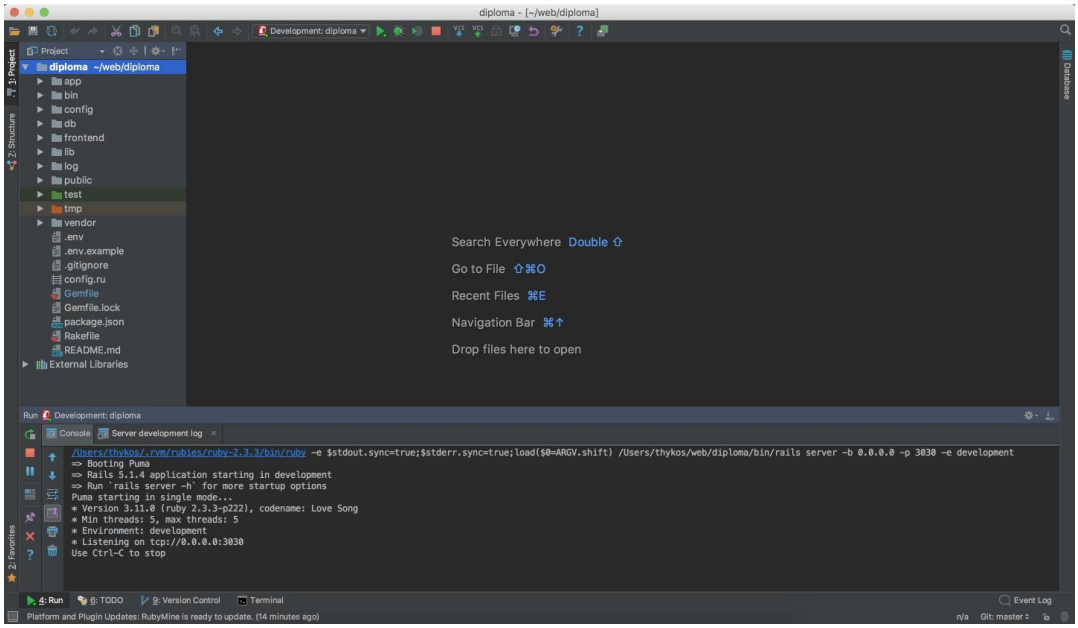



Рисунок 3.1 - Вікно середовища розробки RubyMine

Для початку роботи нам треба ініціалізувати проект. Щоб це зробити достатньо прописати в терміналі ‘rails new app’. Приклад цієї дії зображено на рисунку 3.2 та результат на рисунку 3.3.

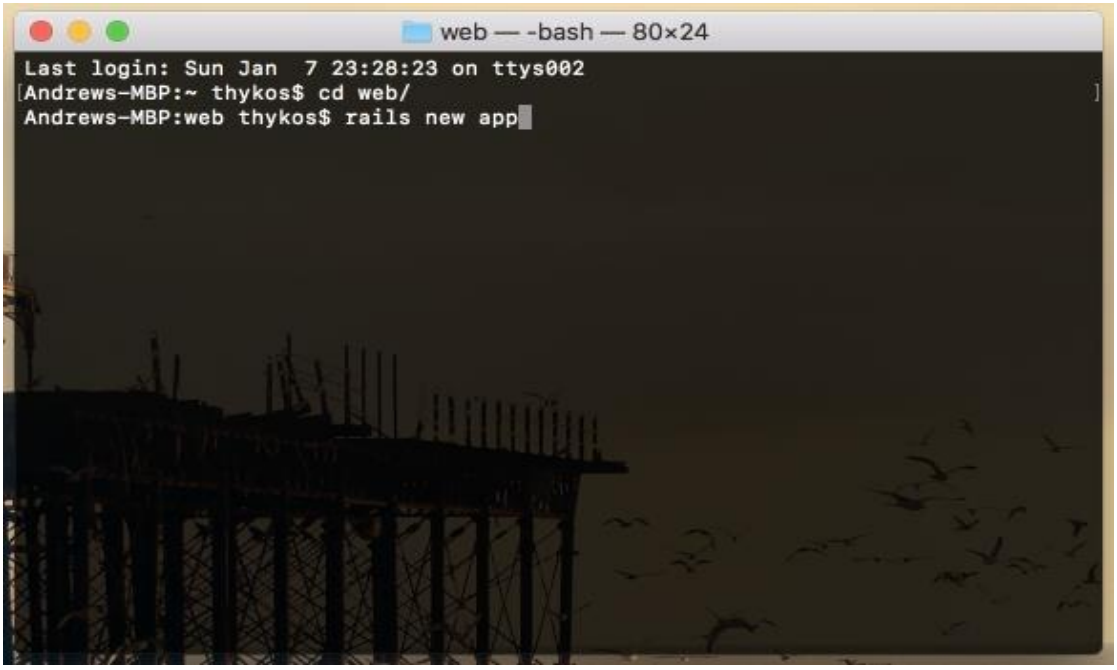



Рисунок 3.2 - Ініціалізація проекту за допомогою терміналу

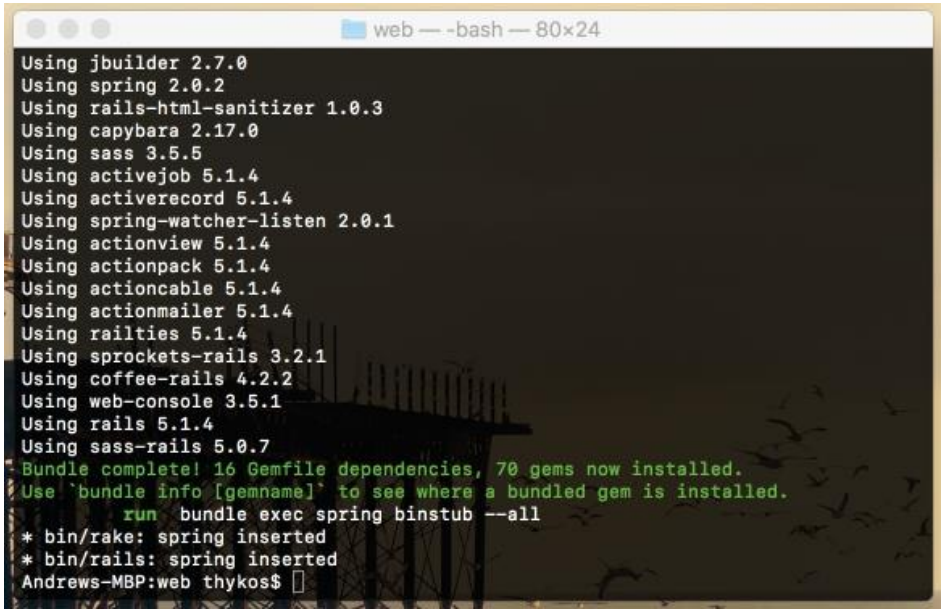



Рисунок 3.3 - Приклад успішного результату ініціалізації проекту.

Перед тим, як почати писати основний функіцонал системи, нам потрібно встановити необхідні нам пакета. В Ruby вони називаються “gem”. На рисунку 3.4 відображено вміст файлу “Gemfile’, який є списком пакетів для нашого додатку.

```
source 'https://rubygems.org'

git_source( name :github) do |repo_name|
  repo_name = "#{repo_name}/#{repo_name}" unless repo_name.include?("/")
  "https://github.com/#{repo_name}.git"
end

gem 'rails', '~> 5.1.4'
gem 'pg', '~> 0.18'
gem 'puma', '~> 3.7'
gem 'sass-rails', '~> 5.0'
gem 'uglifier', '>= 1.3.0'
gem 'coffee-rails', '~> 4.2'
gem 'turbolinks', '~> 5'
gem 'jbuilder', '~> 2.5'

group :development, :test do
  gem 'byebug', platforms: [:mri, :mingw, :x64_mingw]
  gem 'capybara', '~> 2.13'
  gem 'selenium-webdriver'
end

group :development do
  gem 'web-console', '>= 3.3.0'
  gem 'listen', '>= 3.0.5', '< 3.2'
  gem 'spring'
  gem 'spring-watcher-listen', '~> 2.0.0'
end

gem 'tzinfo-data', platforms: [:mingw, :mswin, :x64_mingw, :jruby]
# custom gems
gem 'dotenv-rails'
gem 'devise'
gem 'devise_token_auth'
gem 'omniauth'
gem 'kaminari'
gem 'ransack'
gem 'active_model_serializers', '~> 0.10.0'
gem 'pundit'
gem 'validates_timeliness', '~> 4.0'
```

Рисунок 3.4 - Вміст файлу Gemfile

Всі пакети, окрім тих, що прописані під коментарем “# custom gems”, є стандартними для фреймворку Ruby on Rails, тому їх ми не будемо детально описувати. Для розробки нам необхідні наступні сторонні пакети:

– dotenv-rails

– devise

– devise_token_auth
– omniauth
– kaminari
– ransack
– active_model_serializers
– pundit
– validates_timeliness

Звернемо більш детальну увагу до кожного пакету.

“dotenv-rails” - даний пакет застосовується для зручної розробки та запуску додатку в різних видах середовищах: development, test, production.

“devise”, “devise_token_auth” та “omniauth” - ці три пакети взаємопов'язані. Вони використовуються для налаштування моделі користувача, в основному для авторизації та аутенфікації користувача в системі. За допомогою даних пакетів ми можемо швидко та зручно налаштувати регістрацію, авторизацію користувача. Надати можливість відновлювати пароль за допомогою пошти.

“kaminari” - даний пакет використовуються для пагінації записів при запиті зі сторони клієнта.

“ransack” - цей дуже популярний “gem” служить для зручного пошуку даних в базі даних за вказаними полями, для фільтрації та сортування даних.

“active_model_serializers” - додатковий пакет для ORM-системи Active Record. З його допомогою ми можемо конструювати структуру відповіді на запити зі сторони клієнта. Таким чином, користувач буде отримувати лише ті дані, та в тому виді, в якому це прописано в системі. Це служить для захисту системи.

“pundit” - за допомогою цього пакету, ми можемо настроювати рівні доступу користувачів до певних методів контролерів, та фільтрувати дані, які користувач відсилає для створення чи оновлення моделі.

“validates_timeliness” - цей невеликий пакет служить для валідації даних типу Datetime.

На цьому етапі ми можемо розпочинати розробку нашого додатку. Наша система спроектована за методологією MVC. Але, так як, ми будемо використовувати SPA-додаток на стороні клієнта, ми переналаштували наш сервер на роботу з API, замість звичайного рендеру представлення(view). API дозволяє робити запит до методів контролерів за допомогою спеціальних шляхів.

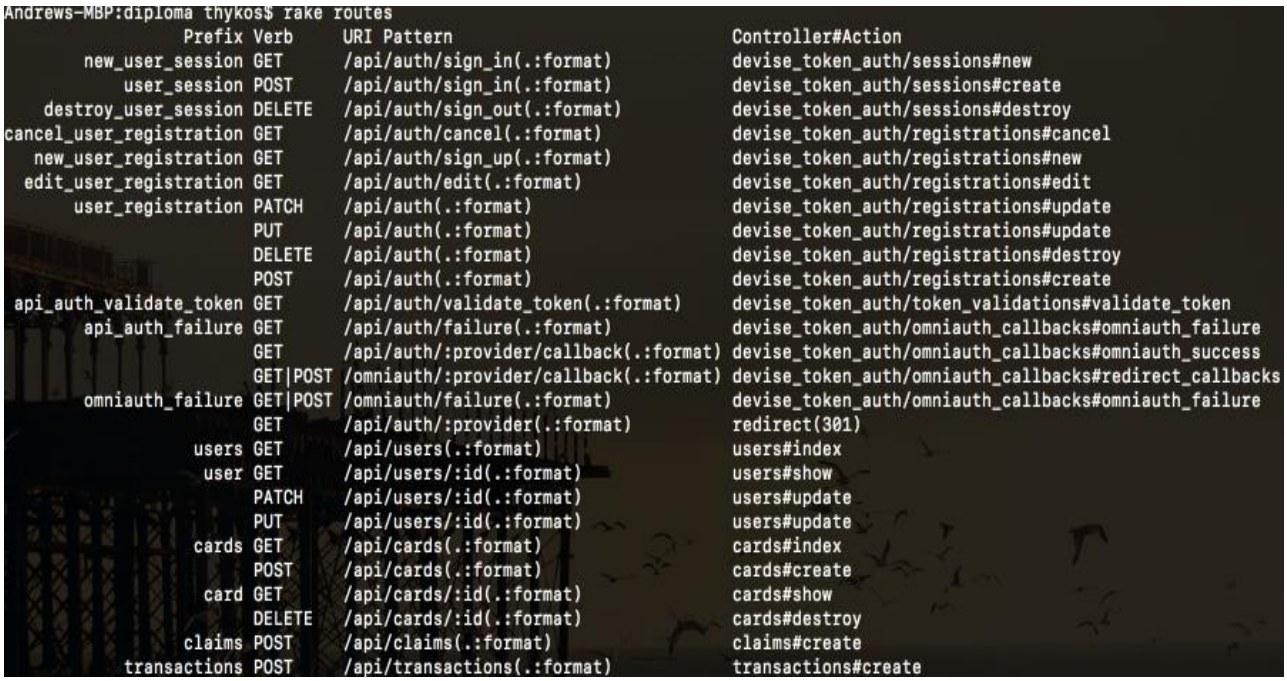
Кожен модуль нашого додатку складається з моделі та контролеру. Модель використовується для зв’язку з базою даних за допомогою ORM-системи. Щоб додати певну таблицю в базу, нам необхідно зробити міграцію. Міграція використовується для додавання, видалення чи зміни структури таблиці в базі даних без нанесення шкоди додатку та запобігає втраті даних.

Опираючись на структуру нашої бази даних, детально описану вище, ми створюємо наступні моделі: User, Card, Transaction, Account, Claim, Blacklist.

Для взаємодії клієнтського інтерфейсу з системою нам необхідно додати наступні контролери: CardsController, UsersController, ClaimsController та TransactionsController. Та прописати створити маршрутизатор для них. Приклад коду маршрутизатора додатка показано на рисунку 3.5, а список маршрутів  для взаємодії з клієнтським додатком на рисунку 3.6.

```
1  Rails.application.routes.draw do
2    scope :api do
3      mount_devise_token_auth_for resource 'User', opts at: 'auth'
4      resources :users, only: [:index, :show, :update]
5      resources :cards, only: [:index, :show, :destroy, :create]
6      resources :claims, only: [:create]
7      resources :transactions, only: [:create]
8    end
9  end
```

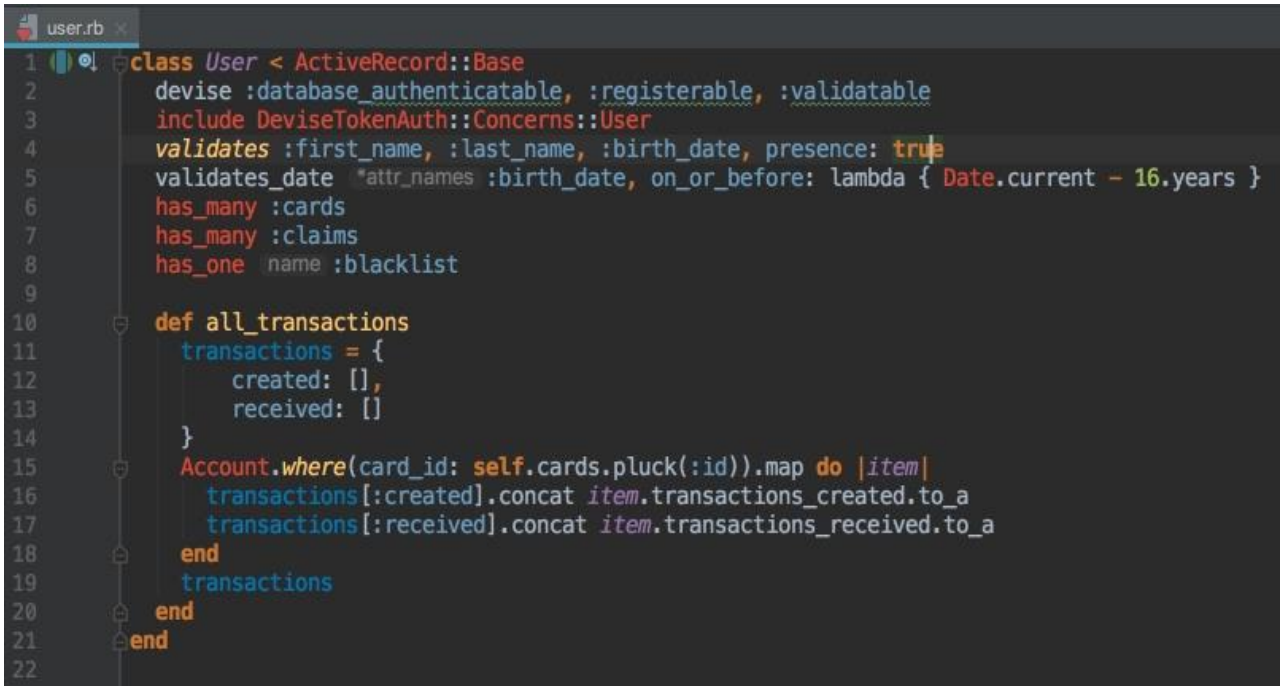
Рисунок 3.5 - Код маршрутизатора додатку



```
Andrews-MBP:diploma thykos$ rake routes
                Prefix Verb     URI Pattern                                Controller#Action
      new_user_session GET      /api/auth/sign_in(.:format)                devise_token_auth/sessions#new
          user_session POST     /api/auth/sign_in(.:format)                devise_token_auth/sessions#create
  destroy_user_session DELETE   /api/auth/sign_out(.:format)               devise_token_auth/sessions#destroy
cancel_user_registration GET    /api/auth/cancel(.:format)                 devise_token_auth/registrations#cancel
  new_user_registration GET     /api/auth/sign_up(.:format)                devise_token_auth/registrations#new
 edit_user_registration GET     /api/auth/edit(.:format)                   devise_token_auth/registrations#edit
     user_registration PATCH    /api/auth(.:format)                        devise_token_auth/registrations#update
                       PUT      /api/auth(.:format)                        devise_token_auth/registrations#update
                       DELETE   /api/auth(.:format)                        devise_token_auth/registrations#destroy
                       POST     /api/auth(.:format)                        devise_token_auth/registrations#create
api_auth_validate_token GET     /api/auth/validate_token(.:format)         devise_token_auth/token_validations#validate_token
      api_auth_failure GET      /api/auth/failure(.:format)                devise_token_auth/omniauth_callbacks#omniauth_failure
                       GET      /api/auth/:provider/callback(.:format)     devise_token_auth/omniauth_callbacks#omniauth_success
                       GET|POST /omniauth/:provider/callback(.:format)     devise_token_auth/omniauth_callbacks#redirect_callbacks
     omniauth_failure GET|POST /omniauth/failure(.:format)                 devise_token_auth/omniauth_callbacks#omniauth_failure
                       GET      /api/auth/:provider(.:format)              redirect(301)
                 users GET      /api/users(.:format)                       users#index
                  user GET      /api/users/:id(.:format)                   users#show
                       PATCH    /api/users/:id(.:format)                   users#update
                       PUT      /api/users/:id(.:format)                   users#update
                 cards GET      /api/cards(.:format)                       cards#index
                       POST     /api/cards(.:format)                       cards#create
                  card GET      /api/cards/:id(.:format)                   cards#show
                       DELETE   /api/cards/:id(.:format)                   cards#destroy
                claims POST     /api/claims(.:format)                      claims#create
          transactions POST     /api/transactions(.:format)                transactions#create
```

Рисунок 3.6 - Маршрути системи для взаємодії з клієнтським додатком

Отже, тепер ми можемо детально розглянути кожну модель нашої системи. На рисунку 3.7 зображено код моделі User.



```ruby
class User < ActiveRecord::Base
  devise :database_authenticatable, :registerable, :validatable
  include DeviseTokenAuth::Concerns::User
  validates :first_name, :last_name, :birth_date, presence: true
  validates_date *attr_names :birth_date, on_or_before: lambda { Date.current - 16.years }
  has_many :cards
  has_many :claims
  has_one  name :blacklist

  def all_transactions
    transactions = {
        created: [],
        received: []
    }
    Account.where(card_id: self.cards.pluck(:id)).map do |item|
      transactions[:created].concat item.transactions_created.to_a
      transactions[:received].concat item.transactions_received.to_a
    end
    transactions
  end
end
```

Рисунок 3.7 - Код моделі User

Існує декілька видів зв’язку: has_one, has_many, belongs_to, has_many :through, has_one :through, has_and_belongs_to_many. В даній моделі описані зв’язки даної таблиці з іншими:

1. Cards - has_many;
2. Claims - has_many;
3. Blacklist - has_one.

Поля first_name, last_name та birth_date провалідовані, як необхідні. А birth_date, яке зберігає дату народження користувача повинно дорівнювати або бути більшим ніж 16 років, адже за чинним законодавством України банківськими картками може користуватись юридична чи фізична особа, яка досягла 16 років.

Медот all_transactions використовується для доступу контроллером до всіх транакцій користувача, як створених, так й отриманих.

Згідно з необхідними нам маршрутами для клієнтської частини ми розробили контролер для моделі User. Код представлений на рисунку 3.8.

```
class UsersController < ApplicationController
  before_action :authenticate_user!

  def index
    users = User.all
    render_resources resources users, options each_serializer: UserPublicSerializer
  end

  def update
    user = User.find( *ids params[:id])
    user.update( attributes permitted_attributes( record User))
    render_resource_or_errors resource user, options serializer: :serializer
  end

  def show
    user = User.includes(:cards).find(params[:id])
    render_resource_or_errors resource user, options serializer: :serializer
  end

  def serializer
    if current_user.id == params[:id]; UserSerializer else UserPublicSerializer end
  end
end
```

Рисунок 3.8 - Код контролеру UsersController

В даному контролері ми добавили метод serializer, який в залежності від користувача, обирає який об’єм даних віддавати користувачу. Якщо користувач посилає запит на свій профайл, він отримає всі дані, що є в базі на нього, включаючи картки. В іншому випадку, він отримає лише публічні дані(ім’я, прізвище та дату народження).

В контролері CardsController описані методи взаємодії користувача з моделлю карток. Як ми бачимо на рисунку 3.9 - клієнт може отримати список всіх своїх карток, разом з їх рахунками, створити нову чи видалити.

```
class CardsController < ApplicationController
  before_action :authenticate_user!, only: [:show, :index, :destroy]
  def show
    card = Card.find( *ids params[:id])
    render_resource_or_errors  resource card,  options serializer: CardFullSerializer
  end

  def destroy
    current_user.cards.where(id: params[:id]).destroy_all
    render json: {status: 'ok' }
  end

  def create
    card = Card.new(permitted_attributes( record Card))
    card.save
    render_resource_or_errors  resource card,  options serializer: CardFullSerializer
  end

  def index
    cards = Card.where(user_id: current_user.id).includes(:account)
    render json: { resources: cards }, each_serializer: CardFullSerializer
  end
end
```

Рисунок 3.9 - Код контролеру CardsController

Згідно з питань безпеки ми добавили валідацію картки. Номер картки перевіряється за допомогою алгоритму Луна, який використовується всіма банками для створення унікального номеру. Приклад даної моделі та методу перевірки представлено на рисунку 3.10. Щоб користувач не мав доступу до інших карток, ми створили policy (рис 3.11), який відповідає за рівні доступу

до моделі. Згідно коду CardPolicy, доступ до дій з моделлю картки має лише її власник, що забезпечує безпеку системи.

```ruby
class Card < ActiveRecord::Base
  validates :cvv2, :expiry_date, :user_id, presence: true
  validate :check_card_nubmer_by_luhn?, on: [:create, :update]
  validates_length_of *attr_names :number, :minimum => 16, :maximum => 16, :allow_blank => false
  after_create :create_account
  has_one name :user
  has_one name :account

  def check_card_nubmer_by_luhn?
    digits = number.chars.map(&:to_i)
    check = digits.pop

    sum = digits.reverse.each_slice(2).flat_map do |x, y|
      [(x * 2).divmod(10), y || 0]
    end.flatten.inject(:+)

    valid_card = check.zero? ? sum % 10 == 0 : (10 - sum % 10) == check
    if valid_card
      true
    else
      errors.add(:number, "invalid card number")
    end
  end

  def create_account
    Account.create( attributes card_id: self.id, amount: 0)
  end
end
```

Рисунок 3.10 - Код моделі Card

```ruby
class CardPolicy < ApplicationPolicy
  def show?
    record.user_id === user.id
  end

  def index?
    show?
  end

  def create?
    show?
  end

  def destroy
    show?
  end

  def permitted_attributes
    [:number, :cvv2, :expiry_date, :id, :user_id]
  end

end
```

Рисунок 3.11 - Код CardPolicy

Найважливішою моделлю даної система є Transaction (рисунок 3.12). Саме ця частина проекту створює транзакції та перевіряє чи надійний одержувач коштів при електронному переказі.

```ruby
class Transaction < ActiveRecord::Base
  validates :amount, :account_from_id, :account_to_id, :result, presence: true
  enum result: [:approved, :declined]
  before_save :get_result
  after_save :change_amount

  def change_amount
    if self.result === 0
      acc_from = Account.find( *ids self.account_from_id)
      acc_to = Account.find( *ids self.account_to_id)
      acc_from.update( attributes amount: acc_from.amount - self.amount)
      acc_to.update( attributes amount: acc_to.amount + self.amount)
    end
  end

  def get_result
    user_to = Account.find( *ids self.account_to_id).card.user_id
    if Blacklist.where(user_id: user_to).size > 0
      self.result = 1
    else
      check_locations
    end
  end


  def check_locations
    transactions_list = Transaction.where(account_from_id: self.account_from_id)
    cities = transactions_list.pluck( *column_names :city)
    countries = transactions_list.pluck( *column_names :country)
    if cities.last != self.city && countries.last != self.country
      self.result = 1
    else
      self.result = 0
    end
  end
end
```

Рисунок 3.12 - Код моделі Transaction

Дана модель містить два методи перевірки надійності одержувача. В першому перевіряється чи є користувач-одержувач в чорному списку. Другий - звіряє місцеположення останньої створеної транзакції користувачем, що намагається провести дану операцію, з нинішнім.

На рисунку 3.13 зображено код контролеру TransactionsController. Згідно з фукнціоналом даного файлу, користувач може створити транзакцію

чи отримати список всіх екземлярів даної моделі лише якщо він авторизований в системі, і лише до тих, що належать йому (отримані чи відправлені).

```
class TransactionsController < ApplicationController
  before_action :authenticate_user!

  def create
    from = Account.where(card_id: params[:resource][:card_id]).first
    card_to = Card.where(number: params[:resource][:card_to_number]).first.try( *a :id)
    to_id = Account.where(card_id: card_to).first.try( *a :id)
    if to_id
      transaction = Transaction.create( attributes {
                                          amount: params[:resource][:amount],
                                          account_from_id: from.id,
                                          account_to_id: to_id,
                                          loc: params[:resource][:loc],
                                          city: params[:resource][:city],
                                          country: params[:resource][:country],
                                          ip: params[:resource][:ip]
                                        })
      transaction.save
      render_resource_or_errors  resource transaction
    else
      render json: {errors: {account_to_id: 'Такой карты нет в системе'}}
    end

  end

  def index
    total = current_user.all_transactions[:created].size + current_user.all_transactions[:received].size
    render json: {resources: current_user.all_transactions, meta: {total: total}}
  end
end
```

Рисунок 3.13 - Код контролеру TransactionsContrloller

Вище вже неодноразово згадувалось про чорний список. Код даної моделі представлено на рисунку 3.14. Цей функціонал базується на зв’язках з моделями Claims та User. Дана модель не має контролера, тому що її екземпляри створюються за допомогою callback-функції after_create моделі Claims.

```
class Blacklist < ActiveRecord::Base
  has_many :claims
  belongs_to :user
end
```

Рисунок 3.14 - Код моделі Blacklist

Модель Claim служить для зберігання скарг в базі даних. Згідно з алгоритмом нашої системи, якщо на користувача поступає більш ніж 3 скарги за шахрайство, він автоматично потрапляю до чорного списку, і не зможу більше створювати транзакції. Таким чином додаток перешкоджає шахрайським операціям клієнтів системи. Більш детально це можна переглянути на рисунку 3.15 з кодом даної моделі.

```
class Claim < ActiveRecord::Base
  validates :description, :user_id, presence: true
  after_create :add_user_to_black_list
  belongs_to :user

  def add_user_to_black_list
    claims = Claim.where(user_id: user_id)
    if claims.size > 3
      Blacklist.create( attributes user_id: user_id, claim_ids: claims.grep(:id) )
    end
  end
end
```

Рисунок 3.15 - Код моделі Claim

Створювати скарги може лише авторизований в системі користувач. Як і отримувати список всіх створених ним скарг. Код цього контролера зображено на рисунку 3.16.

```
class ClaimsController < ApplicationController
  before_action :authenticate_user!
  def create
    claim = Claim.create( attributes permitted_attributes( record Claim))
    claim.save
    render_resource_or_errors  resource claim
  end

  def index
    claims = Claim.where(author: current_user.id)
    render json: {resources: claims, meta: { total: claims.size }}
  end
end
```

Рисунок 3.16 - Код контролера ClaimsController

Останньою моделю даної системи є Account, тобто рахунок, який прив’язаний до картки. Зважаючи на те, що цей додаток - прототип - у кожної окремої картки окремий рахунок. Код цієї моделі зображено на рисунку 3.17.

```
class Account < ActiveRecord::Base
  validates :amount, presence: true
  belongs_to :card

  def transactions_created
    Transaction.where(account_from_id: self.id)
  end

  def transactions_received
    Transaction.where(account_to_id: self.id)
  end

  def transactions
    transactions_created.to_a.concat transactions_received.to_a
  end
end
```

Рисунок 3.17 - Код моделі Account

Отже, зважаючи на всі перечислені моделі та контролери, ми маємо повноцінний сервер для нашої системи.

## 3.2 Розробка клієнтського інтерфейсу додатку

Для розробки клієнтського інтерфейсу було вирішено використовувати HTML, CSS та Javascript, так як наш додаток - це веб-сайт. Щоб розробити інтелектуальний інтерфейс, нам необхідно задіяти спеціалізовану бібліотеку ReactJS. Це дозволить додатку працювати без перезавантаження при кожному

переході по посиланням внутрішнього назначення. Такий тип додатку називається - SPA(Single Page Application).

Як середовище розробки було вирішено використовувати WebStorm. JetBrains WebStorm - інтегрована середовище розробки на JavaScript, CSS & HTML від компанії JetBrains, розроблена на базі платформи IntelliJ IDEA. Ця програма забезпечує автодоповнення, аналіз коду на літ, навігацію по коду, рефакторинг, відладку та інтеграцію з системами керування версіями. Важливим перевагою інтегрованої середовища розробки WebStorm є робота з проектами (у тому числі, рефакторинг JavaScript коду, що знаходиться в різних файлах і папках проекту, а також вкладений в HTML). Підтримується множинна вкладення (коли в документі HTML вкладений сценарій на Javascript, у який вкладено інший код HTML, в якому вкладено Javascript).

Для зручності розробки ми використовуємо готове рішення до архітектури проекту React-create-app, яке базується на Webpack. Так як наш додаток буде писатися на останній версії Javascript(ES7), нам потрібно використовувати бібліотеку “Babel”, щоб він виконувався у всіх браузерах(навіть Internet Explorer від 10 версії). Для розробки нам необхідно щоб наша системи працювала в режимі розробки(для зручності), та компілювалася для роботи на віддаленому сервері. Приведемо список найнеобхідніших залежностей системи, які встановлюються за допомогою пакету npm(Node Package Manager):

- lodash;
- momentJS;
- react;
- react-bootstrap;
- react-modal;
- react-redux;
- react-router;

– redux;

– redux-form;

– superagent.

“lodash” - дана бібліотека містить набір функцій для зручної роботи з різними типами даними: об’єктами, масивами, строками, датами, числами та логічними значеннями.

“momentJS” - бібліотека для зручної роботи з часом.

“react” - бібліотека для роботи з представленнями.

“react-bootstrap” - набір компонент для ReactJS, які базуються на фреймворку Bootstrap.

“react-modal” - компонента модального вікна для ReactJS.

“react-redux” - бібліотека-обгортка для роботи React з Redux.

“react-router” - компонента для роботи зі шляхами в системі. Дозволяє змінювати представлення в залежності від шляху в браузері.

“redux” - рішення для зберігання даних на клієнті. Іншими словами це “база даних” для додатку.

“redux-form” - компонента для зручної роботи з формами додатку. Форми використовуються для заповнення даних для відправлення серверу.

“superagent” - обгортка для методу XHR, який служить для створення запитів. Дозволяє зручно робити запити з використанням promise.

Отже коли ми визначились зі списком залежностей ми можемо розпочати розробляти інтерфейс для додатку. Спочатку треба створити головний файл index.js. В ньому ми підключаємо до додатку “react-router”, “redux” та виводим головну компоненту на сторінку. Код файлу представлено на рисунку 3.18.

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import Root from './containers/Root/Root';
import client from './helpers/ApiClient';
import store from './reducers/store';
import { BrowserRouter, Route } from 'react-router-dom';
import { Provider } from 'react-redux'
import registerServiceWorker from './registerServiceWorker';

client.setStore(store);

ReactDOM.render(
  <Provider store={store}>
    <BrowserRouter>
      <Route component={Root}/>
    </BrowserRouter>
  </Provider>,
  document.getElementById('root'));
registerServiceWorker();
```

Рисунок 3.18 - Код головного файлу index.js

Коли наш додаток відображає на сторінці головну компоненту, користувач потрапляє на сторінку авторизації (рисунок 3.19). Якщо в нього немає облікового запису - він може створити його (рисунок 3.20).

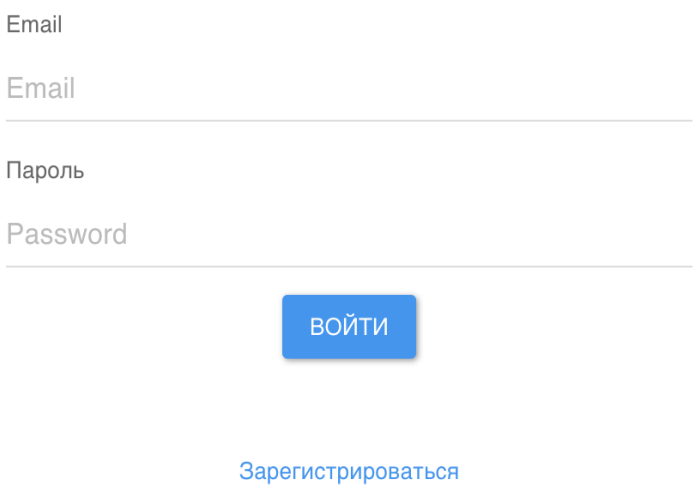



Рисунок 3.19 - Форма авторизації в системі

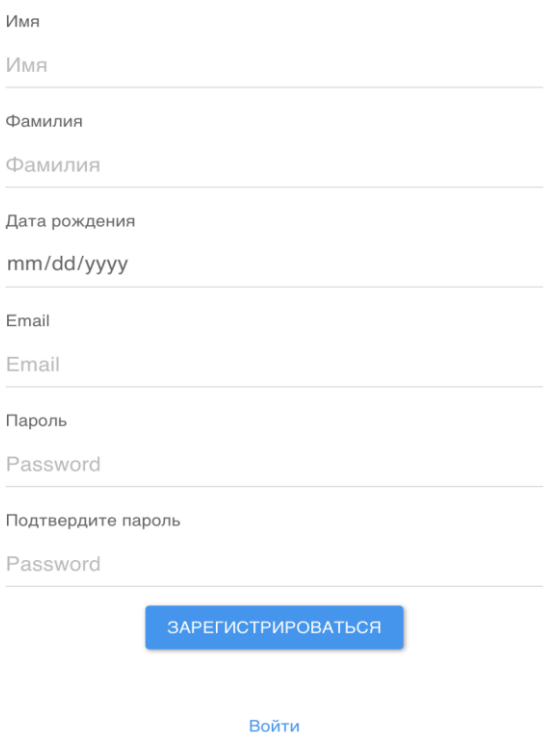



Рисунок 3.20 - Форма регістрації в системі

Після створення облікового запису чи авторизації в системі, додаток зберігає основні дані для авторизації в куках браузера. Це зроблено для того, щоб користувачу не доводилося постійно вводити дані для авторизації після перезавантаження сторінки та для підтвердження доступу до серверу при запитах. Після входу в систему користувач потрапляє на головну сторінку.

Представлення головної сторінки залежить від даних користувача. Якщо він вже додав хоча б одну картку, він побачить інтерфейс як на рисунку 3.21, в іншому випадку як на рисунку 3.22.

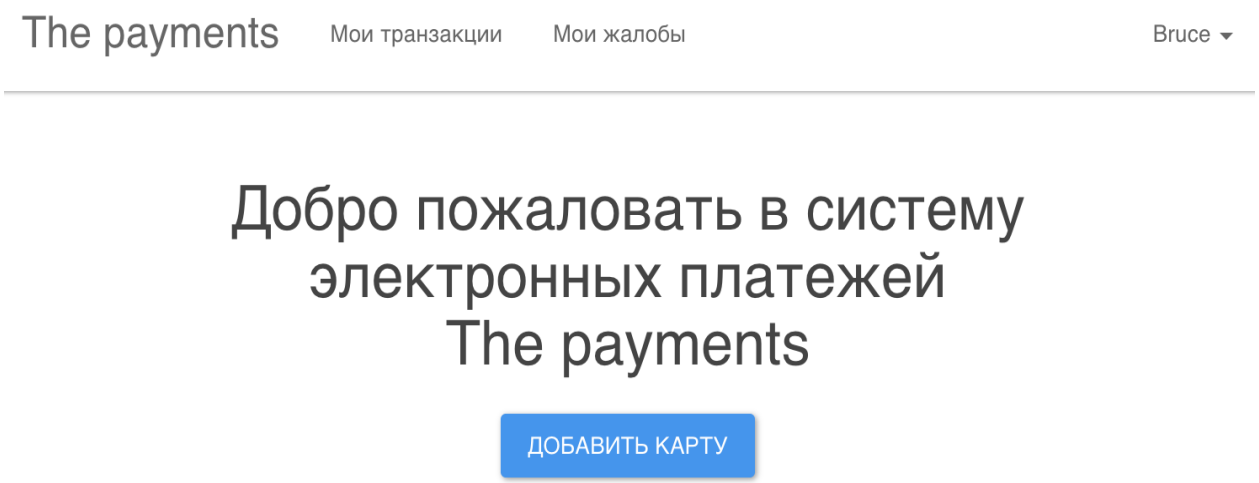



Рисунок 3.21 - Головна сторінка для користувача без карток

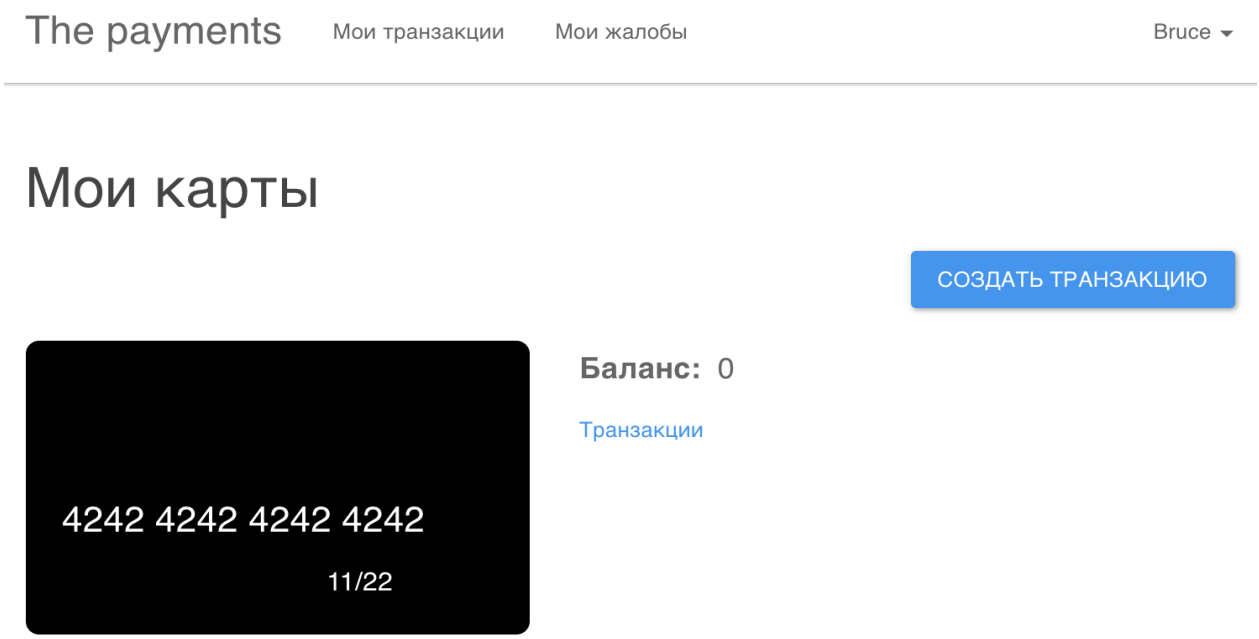



Рисунок 3.22 - Головна сторінка для користувача з картками

Щоб додати картку з головної сторінки на ній створена кнопка “Додати картку”, яка викликає модальне вікно (рисунок 3.23) для додання картки в базу даних серверу.

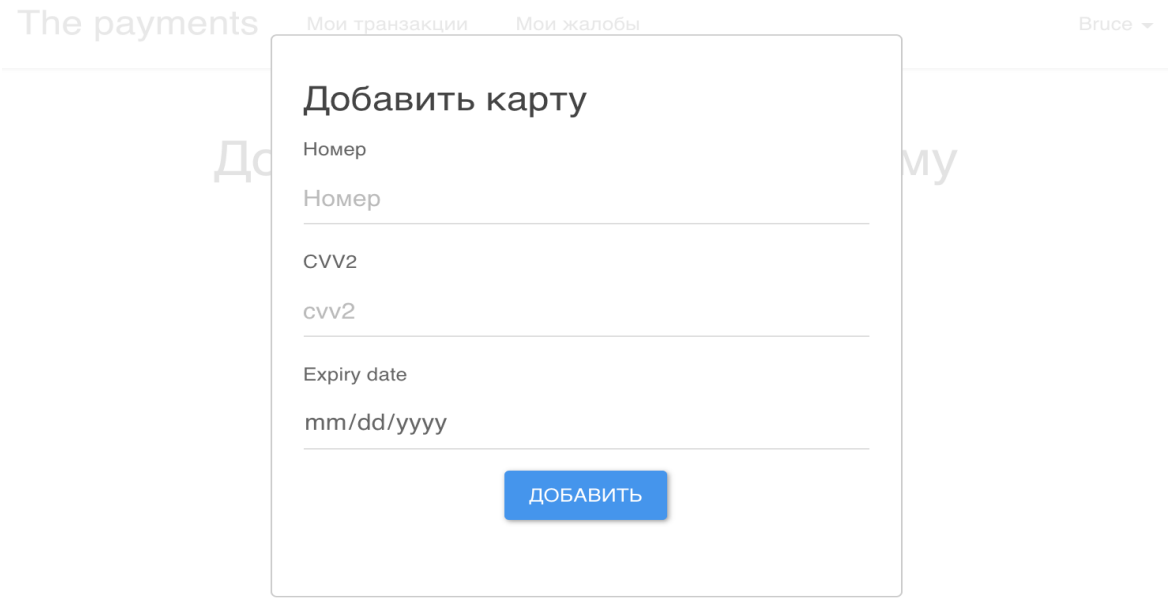



Рисунок 3.23 - Модальне вікно додавання картки

Після створення картки, дані про прив’язаний до неї рахунок підгрузяться з серверу і користувач зможе створити транзакцію за допомогою спеціального модального вікна (рисунок 3.24). Якщо картки отримувача не існує в базі даних серверу - відобразиться сповіщення “Дана картка в системі відсутня” (рисунок 3.25).

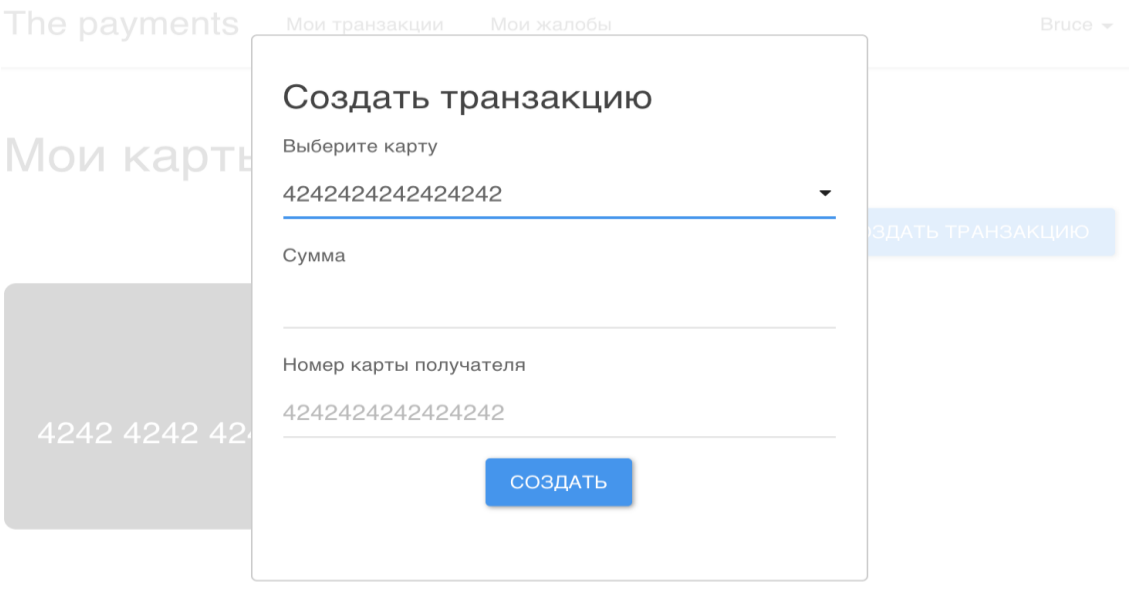



Рисунок 2.24 - Модальне вікно створення транзакції

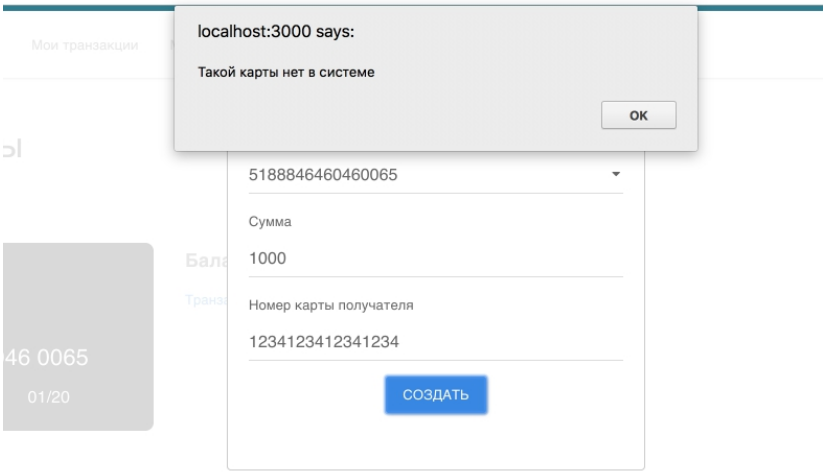



Рисунок 3.25 - Сповіщення про помилку при створенні транзакції

Користувач може змінити свої основні дані у будь-який час за допомогою форми на сторінці налаштунків профілю (рисунок 3.26).

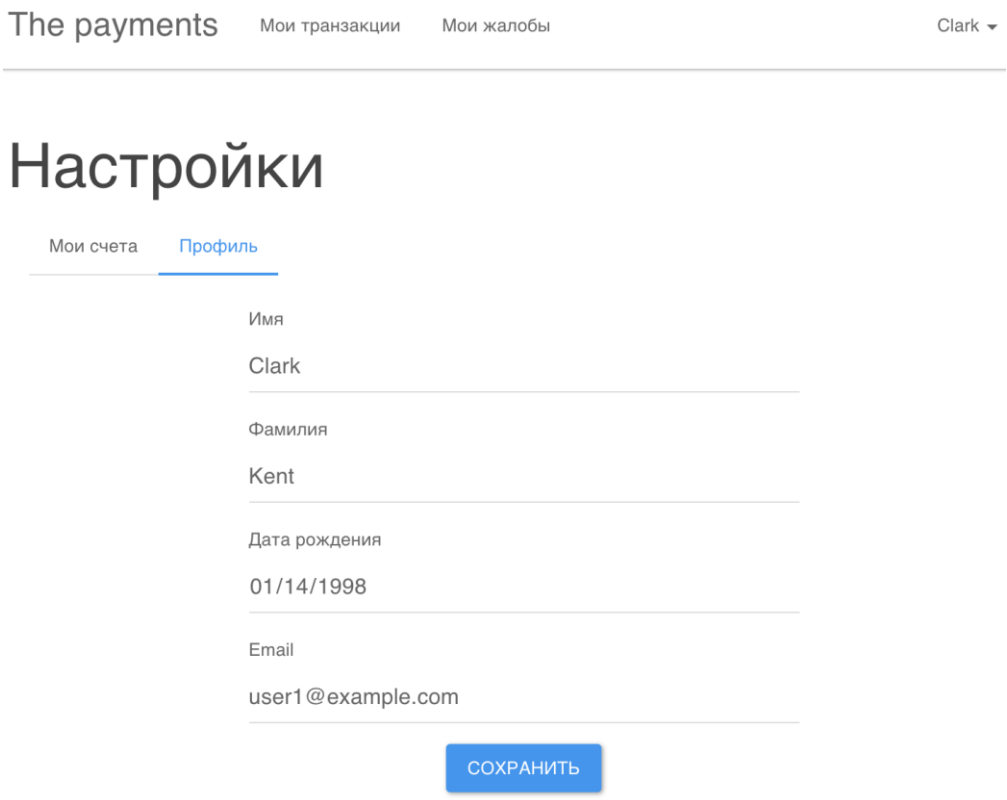



Рисунок 3.26 - Форма даних користувача на сторінці налаштувань

Користувач може переглянути список своїх рахунків та карток на сторінці рахунків в налаштунках (рисунок 3.27). На даному відображені також можна видалити картку з рахунком чи додати нову.

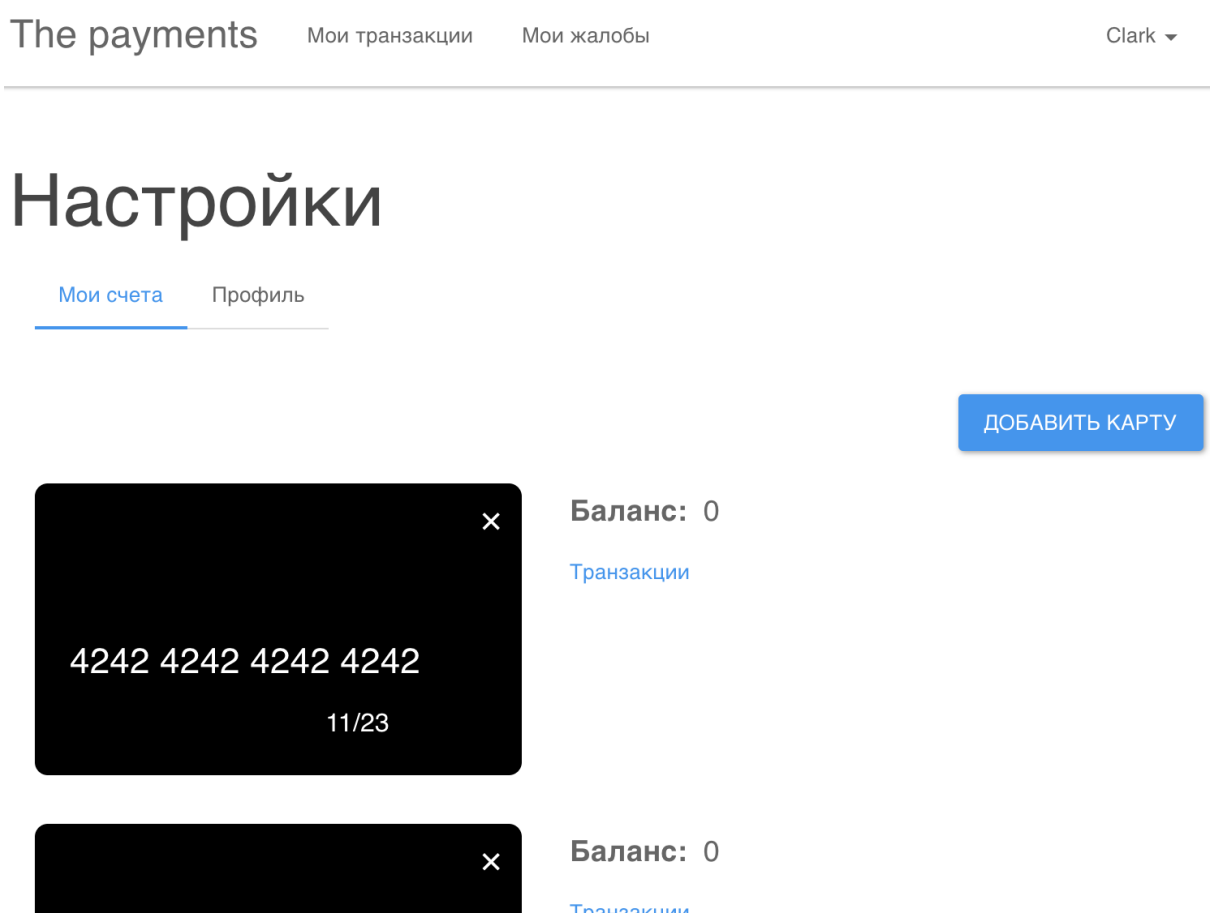



Рисунок 3.27 - Сторінка зі списком рахунків користувача

Біля кожної картки є посилання на її сторінку, де відображена детальна інформація по здійсненим транзакціям з даної картки. Ця сторінка зображена на рисунку 3.28.

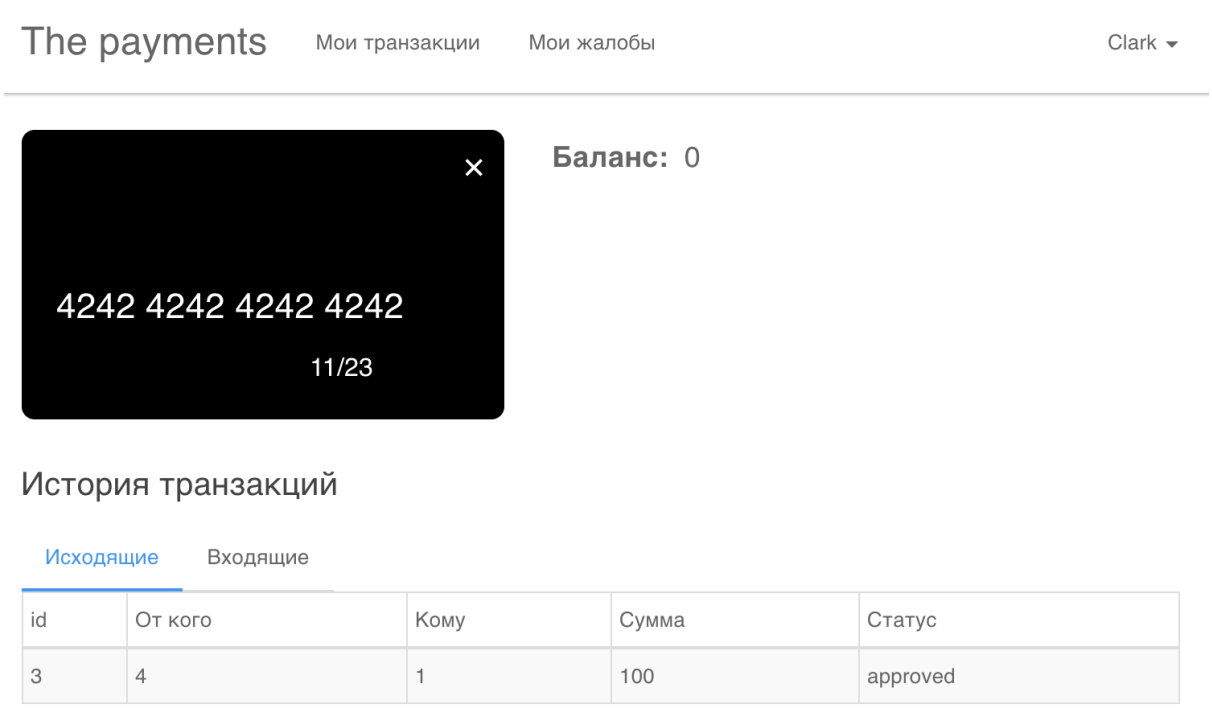



Рисунок 3.28 - Сторінка картки з транзакціями

Користувач може переглянути всі транзакції з усіх карток на окремій сторінці транзакцій, на яку можна потрапити натиснувши на посилання в навігації сайту. Дана сторінка зображена на рисунку 3.29.

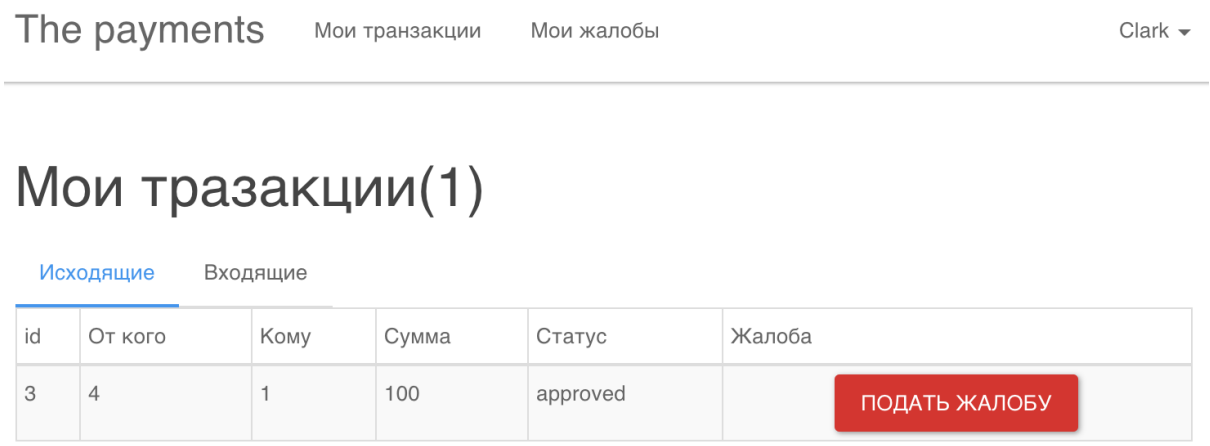



Рисунок 3.29 - Сторінка усіх транзакцій

У останньому рядку таблиці транзакцій створена кнопка для створення скарг. Натиснувши на неї з’явиться модальне вікно з відповідною формою (рисунок 3.30).

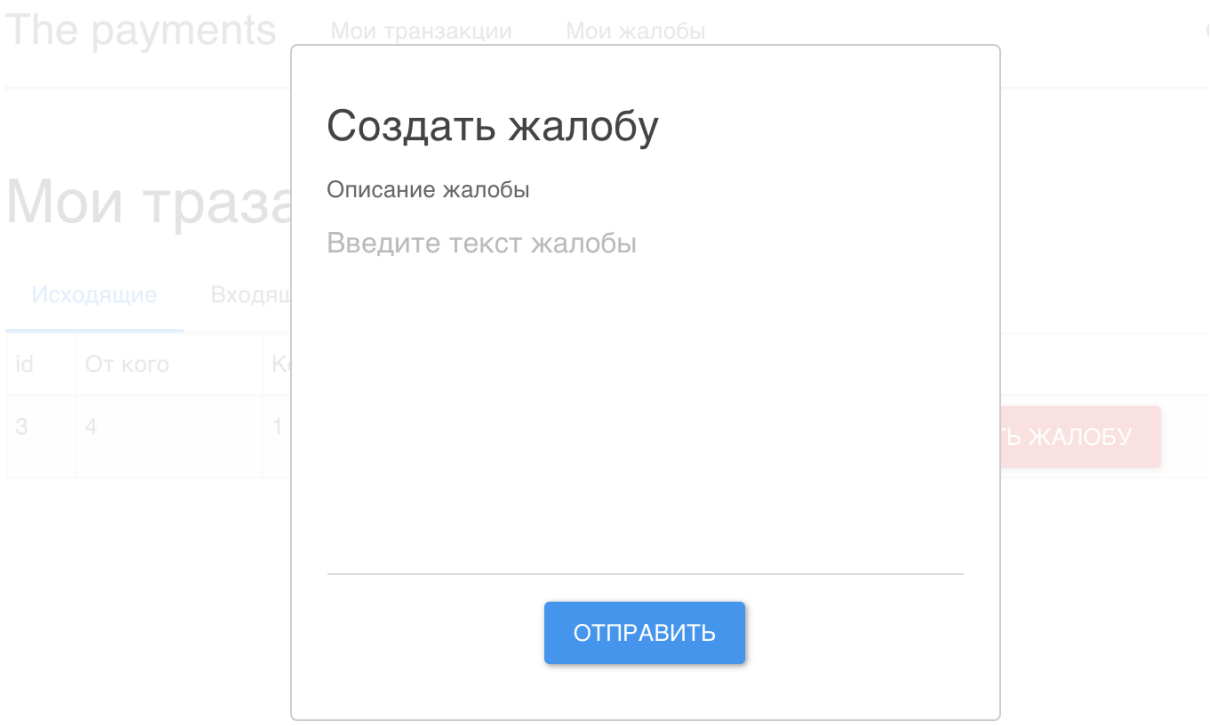



Рисунок 3.30 - Форма створення скарг

Користувач може переглянути увесь список створених ним скарг на окремій сторінці (рисунок 3.31). Список виводиться у вигляді таблиці, в якій міститься ідентифікаційний номер скарги, ім’я та прізвище користувача на котрого вона була створена та її опис.


Рисунок 3.31 - Сторінка зі списком скарг

Детальний код кожної сторінки та компоненту інтерфейсу надано у додатку А.

## 3.3 Оцінка очікуваних ефектів від впровадження системи

Як відомо продуктивність та ефективність від провадження автоматизованої системи визначають порівнянням результатів її роботи і затрат всіх видів ресурсів, необхідних для її створення і розвитку. Вкладення ресурсів у створення програмного продукту розраховуються за формулою 3.1.

$$K = K_1 + K_2 + K_1 \quad (3.1)$$

де $K_1$ - витрати на розробку;

$K_2$ - амортизація ноутбука;

$K_3$ - затрати на електроенергію.

Середня вартість роботи 1 години розробника на ринку становить 20 дол/год, що за нинішнім курсов валют дорівнює 571 грн/год. Розробка даного прототипу склала 50 годин. Тобто витрати на розробку склали 28555 грн. Враховуючи 22 відсотки на фонд заробітної плати, помножаємо дану суму на 1.22 та отримуємо, що $K_1$ = 34837.1 (грн).

Вартість ноутбука, за яким створювався даний прототип на ринку складає в середньому 36500 грн. Амортизація технічного засобу вираховується за формулою 3.2

$$K_2 = \frac{C}{4 * 292} * 50 \quad (3.2)$$

де С - вартість ноутбука.

Отже, виходячи з формули 3.2, $K_2$ = 1562.5 грн.

Згідно з інструкцією до ноутбука, він потребляю 40 ватт/год при зарядці. 1 зарядку ноутбука вистачає на 8 годин роботи, а заряджається повністю за 2 години. Отже було використано 6.25 повних зарядів аккамулятора для розробки даного прототипу. Це 1000 ватт/год, або 0.25 кіловатт/год. Ціна за 1 кіловатт/год в Україні на даний момент становить 0.9грн., отже згідно розрахунків $K_3$ = 0.225 (грн).

Згідно формули 3.1 K = 34837.1 + 1562.5 + 0.225 = 36399.825 (грн).

У 2016 році всі банки України втратили 339.18 млн грн через шахрайство. В нашій державі на 2016 рік існувало 98 банків. Допустимо, що всі банки втратили однакову суму від махінацій, тобто 339.18/98 = 3461020 грн.

Мінімальна економія від впровадження розробленної системи становить 1%. Отже економія буде складати 34610.2 грн.

Економічний ефект розраховується за формулою 3.3.

$$E_{\text{р}} = E \ - \ K \ * \ 0.33 \tag{3.3}$$

де Е - економія, К - витрати.

Виходить що $E_{\text{р}}$ = 34610.2 - 36399.825 * 0.33 = 22598.25 (грн).

Термін окупності розраховується за формулою 3.4.

$$T_{\text{р}} = \frac{K}{E_{\text{р}}} \tag{3.4}$$

$T_{\text{р}}$ = 36399.825 / 22598.25 = 1.61 року.

При ефективному використанні ресурсів розрахунковий термін окупності повинен бути менше нормативного $T_{\text{н}}$ = 2.4, у нашому випадку це досягається.

Коефіцієнт ефективності розраховується за формулою 3.5.

$$K_{\text{е}} = \frac{E_{\text{р}}}{K} \tag{3.5}$$

$K_{\text{е}}$ = 22598.25 / 36399.825 = 0.62.

Отже, як бачимо розробка і використання додатку є економічно доцільною, так як річна економія складе 22598.25 грн. за рік, коефіцієнт ефективності складе 62%, а термін окупності системи складає 1.61, тобто проект окупиться трохи більше, ніж за півтора роки.

# ВИСНОВКИ

Під час виконання даної роботи було проведено аналіз діючих в Україні систем електронних платежів та були сформовані вимоги, які необхідні для створення прототипу системи, що буде перешкоджати здійсненню шахрайських операцій клієнтів банку. Даний додаток було розроблено під час проходження практики в ТОВ “Резонанс.НЕТ.

Також нами було обрано архітектуру “клієнт-сервер” та технології, за допомогою яких був побудований прототип платіжної системи.

Враховуючи мету написання даної випускної роботи, а саме розробка прототипу створення прототипу системи, що буде перешкоджати здійсненню шахрайських операцій клієнтів банку, можна стверджувати про її досягнення. У процесі написання роботи були докладно розглянуті наступні питання:

– банківська операція як об’єкт шахрайства;

– системи електронних платежів як інструмент здійснення шахрайських операцій;

– зміст реляційної теорії;

– визначення, призначення та головні переваги використання баз даних для створення інформаційних систем;

– процедурна цілісність даних у вигляді використання збережених процедур та представлень.

У результаті виконання роботи було створено 7 базових таблиць, які повністю задовольняють тим задачам, що мають виконуватися за допомогою даної бази. Був розроблений клієнтський інтерфейс для роботи користувачів з системою. За допомогою даного інтерфейс користувач мав змогу зареєструватися, авторизуватись, змінити дані свого профілю, додавати та видаляти картки, створювати транзакції, переглядати їх та створювати скарги на інших користувачів.

Таким чином, мету, поставлену перед нами, можемо вважати досягнутою, а надане для виконання завдання випускної роботи повністю виконаним.

# ПЕРЕЛІК ПОСИЛАНЬ

1. Астахова, И.Ф. SQL в примерах и задачах [Текст] : учебн. пособие / А.П. Толстоборов, В.М Мельников – Минск : Новое знание, 2002. – 176 с. – ISBN 985-475-004-3.

2. Галузинський, Г.П. Перспективні технологічні засоби оброблення інформації [Текст] : навч.-метод. посібник / І.В. Гордієнко –К. : КНЕУ, 2002. – 280 с. – ISBN 966-574-359-7.

3. Дейт, К.Дж. Введение в системы баз данных [Текст] / англ : [6-е изд.]. – К.-М. : Диалектика, 1998. – 784 с. – ISBN 966-506-124-0.

4. Когаловский, М.Р. Энциклопедия технологий баз данних [Текст]. – М. : Финансы и статистика, 2002. – 800 с. – ISBN 5-279-02276-4.

5. Косенко, Е. Реванш встроенных СУБД. Выбор подхода или подход к выбору? [Текст]. – 2002. – № 4. – С. 50–54.

6. PostgreSQL: Documentation [Електронний ресурс]. – Режим доступа : https://www.postgresql.org/docs/ – Заголовок з екрану.

7. Ruby-Doc.org: Documenting the Ruby Language [Електронний ресурс]. – Режим доступа : http://ruby-doc.org/ – Заголовок з екрану.

8. Ruby on Rails API [Електронний ресурс]. – Режим доступа : http://api.rubyonrails.org/ – Заголовок з екрану.

9. Hello World - React [Електронний ресурс]. – Режим доступа : https://reactjs.org/docs/hello-world.html – Заголовок з екрану.

10. Intruduction - Redux [Електронний ресурс]. – Режим доступа : https://redux.js.org/docs/introduction/ – Заголовок з екрану.

11. Redux Form - API [Електронний ресурс]. – Режим доступа : https://redux-form.com/7.2.0/docs/api/ – Заголовок з екрану.

12. React Router: Declarative Routing for React.js [Електронний ресурс]. – Режим доступа : https://reacttraining.com/react-router/web/ guides/philosophy – Заголовок з екрану.

13. SuperAgent — elegant API for AJAX in Node and browsers [Електронний ресурс]. – Режим доступа : http://visionmedia.github.io/superagent/ – Заголовок з екрану.

14. Introduction [Електронний ресурс]. – Режим доступа : https://webpack.js.org/api/ – Заголовок з екрану.

15. activerecord-hackery/ransack: Object-based searching. [Електронний ресурс]. – Режим доступа : https://github.com/activerecord-hackery/ransack – Заголовок з екрану.

16. rails-api/active_model_serializers: ActiveModel::Serializer implementation and Rails hooks [Електронний ресурс]. – Режим доступа : https://github.com/rails-api/active_model_serializers – Заголовок з екрану.

17. varvet/pundit: Minimal authorization through OO design and pure Ruby classes [Електронний ресурс]. – Режим доступа : https://github.com/varvet/pundit – Заголовок з екрану.

18. adzap/validates_timeliness: Date and time validation plugin for ActiveModel and Rails. Supports multiple ORMs and allows custom date/time formats. [Електронний ресурс]. – Режим доступа : https://github.com/adzap/validates_timeliness – Заголовок з екрану.

19. kaminari/kaminari: A Scope & Engine based, clean, powerful, customizable and sophisticated paginator for Ruby webapps [Електронний ресурс]. – Режим доступа : https://github.com/kaminari/kaminari – Заголовок з екрану.

20. omniauth/omniauth: OmniAuth is a flexible authentication system utilizing Rack middleware. [Електронний ресурс]. – Режим доступа : https://github.com/omniauth/omniauth – Заголовок з екрану.

21. bkeepers/dotenv: A Ruby gem to load environment variables from `.env`. [Електронний ресурс]. – Режим доступа : https://github.com/bkeepers/dotenv – Заголовок з екрану.

22. Lodash Documentation [Електронний ресурс]. – Режим доступа : https://lodash.com/docs/4.17.4 – Заголовок з екрану.

23. Moment.js | Docs [Електронний ресурс]. – Режим доступа : https://momentjs.com/docs/ – Заголовок з екрану.

24. Usage with React · Redux [Електронний ресурс]. – Режим доступа : https://redux.js.org/docs/basics/UsageWithReact.html – Заголовок з екрану.

25. https://react-bootstrap.github.io/getting-started/introduction [Електронний ресурс]. – Режим доступа : https://react-bootstrap.github.io/getting-started/introduction – Заголовок з екрану.

26. Introduction · Bootstrap [Електронний ресурс]. – Режим доступа : http://getbootstrap.com/docs/4.0/getting-started/introduction/ – Заголовок з екрану.

27. Font Awesome, the iconic font and CSS toolkit [Електронний ресурс]. – Режим доступа : http://fontawesome.io/ – Заголовок з екрану.

28. Documentation | Node.js [Електронний ресурс]. – Режим доступа : https://nodejs.org/uk/docs/ – Заголовок з екрану.

29. Token based authentication for Rails JSON APIs. Designed to work with jToker and ng-token-auth. [Електронний ресурс]. – Режим доступа : https://github.com/lynndylanhurley/devise_token_auth – Заголовок з екрану.

30. Кваліфікуючі ознаки шахрайства [Електронний ресурс]. – Режим доступа : http://naub.org.ua/?p=1274 – Заголовок з екрану.

31. Про внесення змін до деяких норм... | від 06.09.2016 № 382 [Електронний ресурс]. – Режим доступа : http://zakon2.rada.gov.ua/laws/show/v0382500-16/paran36#n36 – Заголовок з екрану.

32. API · Babel [Електронний ресурс]. – Режим доступа : https://babeljs.io/docs/usage/api/ – Заголовок з екрану.

33. Free themes for Bootstrap / [Електронний ресурс] – Режим доступу: https://bootswatch.com/ – Заголовок з екрану.

# ДОДАТОК А

## SUMMARY

Kondrakov A. V. Automation of submission of applications for subsidies and preferential allowances. – Masters-level Qualification Thesis. Sumy State University, Sumy, 2017.

The paper investigates the object and tools for carrying out fraudulent operations of bank's clients. An analysis of the main criteria for assessing a banking transaction as fraudulent was conducted. A prototype of the electronic payment system was developed based on the implementation of the algorithm of interfering with carrying out fraudulent operations of bank's clients. The purpose of this study is to develop an algorithm for preventing fraudulent operations of bank's clients.

Key words: electronic payment system, fraud, fraudulent operation, banking operation.

## АНОТАЦІЯ

Кондраков А. В. Розробка автоматизованого рішення для виявлення шахрайських операцій клієнтів банку. – Кваліфікаційна магістерська робота. Сумський державний університет, Суми, 2017 р.

У роботі досліджено об’єкт та інструменти для здійснення шахрайських операцій клієнтів банку. Проведений аналіз основних критеріїв оцінювання банківської операції як шахрайської. Розроблено прототип системи електронних платежів з реалізацією алгоритму перешкоджання здійсненню шахрайських операцій клієнтів банку. Основаною метою цього дослідження є розробка алгоритму для перешкоджання здійснення шахрайських операцій клієнтів банку.

Ключові слова: система електронних платежів, шахрайство, шахрайська операція, банківська операція.

# ДОДАТОК Б

## Коди компонент та файлів для створення інтерфейсу

1) Код компоненти App

```
import React, { Component } from 'react';
import Routes from '../../routes';
import { Navbar, NavItem, Nav, NavDropdown, MenuItem } from 'react-bootstrap';
import './styles.css';
import { connect } from 'react-redux';
import { get } from 'lodash';
import { logout } from '../../reducers/auth';
import { setCards } from '../../reducers/cards';
import { Link } from 'react-router-dom';
import client from '../../helpers/ApiClient';

class App extends Component {

 componentDidMount() {
   client.get('/cards')
     .then(response => this.props.setCards(response.resources))
 }

 render() {
   const { user, history: { push } } = this.props;
   return (
     <div className="App">
       <Navbar>
```

```
            <Navbar.Header>
              <Navbar.Brand>
                <Link to="/">The payments</Link>
              </Navbar.Brand>
            </Navbar.Header>
            <Nav>
              <NavItem eventKey={1} href="#" onClick={() => push('/transactions')}>
                Мои транзакции
              </NavItem>
              <NavItem eventKey={2} href="#" onClick={() => push('/claims')}>
                Мои жалобы
              </NavItem>
            </Nav>
            <Nav pullRight>
              <NavDropdown eventKey={3} title={get(user, 'first_name') || 'Пользователь'} id="basic-nav-dropdown">
                <MenuItem eventKey={3.1} onClick={() => push('/settings/cards')}>Настройки</MenuItem>
                <MenuItem eventKey={3.2} onClick={this.props.logout}>Выйти</MenuItem>
              </NavDropdown>
            </Nav>
          </Navbar>
          <div className="container">
            <Routes/>
          </div>
        </div>
```

```
    );
  }
}

export default connect(state => ({user: state.auth.user}), { logout, setCards
})(App);
```

2) Код компоненти Root

```
import React, { Component } from 'react';
import { connect } from 'react-redux';
import { get } from 'lodash';
import App from '../App/App';
import Auth from '../Auth/Auth';
import client from '../../helpers/ApiClient';
import { clearAuthData } from '../../helpers/authData';
import { setUser } from '../../reducers/auth';
import { setCards } from '../../reducers/cards';
import { Route } from 'react-router-dom';

class Root extends Component {

 componentWillMount() {
   client.get('/auth/validate_token')
     .then(response => {
       this.props.setUser(response.data);
     })
     .catch(() => {
```

```
        const { history } = this.props;
        if (!history.location.pathname.includes('login', 'signup')) {
          this.props.history.push('/login');
        }
        clearAuthData();
      });
    }

    componentWillReceiveProps(nextProps) {
      const { user } = nextProps;
      const { history } = this.props;
      const pathname = history.location.pathname;
      if (get(user, 'id') && (pathname.includes('login') || pathname.includes('signup'))) {
        this.props.history.push('/');
      }
      if (!get(user, 'id') && !(pathname.includes('login') || pathname.includes('signup'))) {
        this.props.history.push('/login');
      }
    }


    render() {
      const { user } = this.props;
      return (
        <div className="App">
          {get(user, 'id')
```

```
            ? <Route component={App}/>
            : <Route component={Auth}/>
          }
        </div>
      );
    }
  }

export default connect(state => ({user: state.auth.user}), { setUser, setCards
})(Root);
```

3) Код компоненти Home

```
import React, { Component } from 'react';
import client from '../../helpers/ApiClient';
import { Button } from 'react-bootstrap';
import './styles.css';
import { connect } from 'react-redux';
import { removeCard, addCard } from '../../reducers/cards';
import AddCardModal from
'../../components/AddCardModal/AddCardModal';
import Card from '../../components/Card/Card';
import { get } from 'lodash';
import CreateTransactionModal from
'../../components/CreateTransactionModal/CreateTransactionModal';

class Home extends Component {
 constructor(props) {
```

```
  super(props);
  this.state = {
    showModal: false,
    showTransactionModal: false
  };
}

onShowModal = () => {
  this.setState({
    showModal: true
  });
};

onShowTransactionModal = () => {
  this.setState({
    showTransactionModal: true
  });
};

onCloseModal = () => {
  this.setState({
    showModal: false
  });
};

onCloseTransactionModal = () => {
  this.setState({
    showTransactionModal: false
```

```
    });
  };

  onCreateCard = (data) => {
    const { user } = this.props;
    client.post('cards', { data: { card: {...data, user_id: user.id } }})
      .then(response => {
        this.props.addCard(response.resource);
        this.onCloseModal();
      });
  };

  onCreateTransaction = (data) => {
    fetch(`${window.location.protocol}//ipinfo.io/json`)
      .then(response => response.json())
      .then(location_data => {
        client.post('transactions', { data: { resource: {...data, ...location_data} }})
          .then(response => {
            if (get(response, 'errors.account_to_id'))
alert(response.errors.account_to_id);
          });
      });
  };

  render() {
    const { showModal, showTransactionModal } = this.state;
    const { cards } = this.props;
    return (
```

```
<div className="App">
 {cards.length < 1 &&
  <div className="dummyWrapper">
   <h3 className="text-center">Добро пожаловать в систему электронных платежей <div>The payments</div></h3>
   <div className="text-center addCardBtnWrapper">
    <Button type="submit" bsStyle="primary" onClick={this.onShowModal}> Добавить карту </Button>
    <AddCardModal isOpen={showModal} onSubmit={this.onCreateCard} onClose={this.onCloseModal} />
   </div>
  </div>
 }
 {cards.length > 0 &&
  <div>
   <h3>Мои карты</h3>
   <div className="text-right addCardBtnWrapper">
    <Button type="submit" bsStyle="primary" onClick={this.onShowTransactionModal}> Создать транзакцию </Button>
   </div>
   {cards.map((card, idx) => <Card key={idx} withLink card={card}/>)}
   <CreateTransactionModal isOpen={showTransactionModal} onSubmit={this.onCreateTransaction} onClose={this.onCloseTransactionModal} />
  </div>
 }
</div>
```

```
  );
 }
}

export default connect(state => ({
 user: state.auth.user,
 cards: state.cards
}), { removeCard, addCard })(Home);
```

4) Код компоненти Auth

```
import React, { Component } from 'react';
import SignIn from '../Auth/SignIn';
import SignUp from '../Auth/SignUp';
import { Route, Switch } from 'react-router-dom';

export default class Auth extends Component {
 render() {
  return (
   <Switch>
    <Route path='/login' component={SignIn}/>
    <Route path='/signup' component={SignUp}/>
   </Switch>
  );
 }
}
```

5) Код компоненти SignIn

```
import React, { Component } from 'react';
import { reduxForm, Field } from 'redux-form';
import { Button } from 'react-bootstrap';
import client from '../../helpers/ApiClient';
import { setUser } from '../../reducers/auth';
import { connect } from 'react-redux';
import { Link } from 'react-router-dom';
import './styles.css';

class SignIn extends Component {

 static Form = reduxForm({
   form: 'loginForm',
   fields: ['email', 'password']
 })(({handleSubmit}) => (
   <div className="formWrapper">
     <form onSubmit={handleSubmit}>
       <div className="form-group">
         <label >Email</label>
         <Field
           name="email"
           className="form-control"
           component="input"
           type="email"
           placeholder="Email"
         />
       </div>
       <div className="form-group">
```

```
        <label>Пароль</label>
        <Field
          name="password"
          className="form-control"
          component="input"
          type="password"
          placeholder="Password"
        />
      </div>
      <div className="text-center">
        <Button type="submit" bsStyle="primary"> Войти </Button>
      </div>
    </form>
  </div>
));

onSubmit = (data) => {
  client.post('/auth/sign_in', { data: data })
    .then(response => {
      this.props.history.push('/');
      this.props.setUser(response.data)
    });
};

render() {
  return (
    <div className="container">
      <SignIn.Form onSubmit={this.onSubmit}/>
```

```
        <div className="text-center">
          <Link to="/signup">Зарегистрироваться</Link>
        </div>
      </div>
    );
  }
}

export default connect(null, { setUser })(SignIn);
```

6) Код компоненти SignUp

```
import React, { Component } from 'react';
import { reduxForm, Field } from 'redux-form';
import { Button } from 'react-bootstrap';
import client from '../../helpers/ApiClient';
import { setUser } from '../../reducers/auth';
import { connect } from 'react-redux';
import { Link } from 'react-router-dom';
import './styles.css';

class SignUp extends Component {

 static Form = reduxForm({
   form: 'signUpForm',
   fields: ['email', 'password', 'first_name', 'last_name', 'birth_date',
'password_confirmation']
 })(({handleSubmit}) => (
```

```
<div className="formWrapper">
  <form onSubmit={handleSubmit}>
    <div className="form-group">
      <label >Имя</label>
      <Field
        name="first_name"
        className="form-control"
        component="input"
        type="text"
        placeholder="Имя"
      />
    </div>
    <div className="form-group">
      <label >Фамилия</label>
      <Field
        name="last_name"
        className="form-control"
        component="input"
        type="text"
        placeholder="Фамилия"
      />
    </div>
    <div className="form-group">
      <label >Дата рождения</label>
      <Field
        name="birth_date"
        className="form-control"
        component="input"
```

```
    type="date"
   />
  </div>
  <div className="form-group">
   <label >Email</label>
   <Field
    name="email"
    className="form-control"
    component="input"
    type="email"
    placeholder="Email"
   />
  </div>
  <div className="form-group">
   <label>Пароль</label>
   <Field
    name="password"
    className="form-control"
    component="input"
    type="password"
    placeholder="Password"
   />
  </div>
  <div className="form-group">
   <label>Подтвердите пароль</label>
   <Field
    name="password_confirmation"
    className="form-control"
```

```
                component="input"
                type="password"
                placeholder="Password"
              />
            </div>
            <div className="text-center">
              <Button type="submit" bsStyle="primary"> Зарегистрироваться
</Button>
            </div>
          </form>
        </div>
      ));

      onSubmit = (data) => {
        client.post('/auth', { data: data })
          .then(response => {
            this.props.setUser(response.data);
            this.props.history.push('/');
          });
      };

      render() {
        return (
          <div className="container">
            <SignUp.Form onSubmit={this.onSubmit}/>
            <div className="text-center">
              <Link to="/login">Войти</Link>
            </div>
```

```
    </div>
  );
 }
}

export default connect(null, { setUser })(SignUp);
```

7) Код компоненти Card

```
import React, { Component } from 'react';
import client from '../../helpers/ApiClient';
import { splitNumber } from '../../helpers/card';
import { Tab, Tabs, Table } from 'react-bootstrap';
import moment from 'moment';

class Card extends Component {
 constructor(props) {
  super(props);
  this.state = {
   card: {}
  };
 }

 componentDidMount() {
  const { match } = this.props;
  client.get(`cards/${match.params.id}`)
   .then(response => this.setState({
    card: response.resource
   }));
```

```
}

renderTable = (data) => {
  return (
    <Table striped bordered condensed hover>
      <thead>
        <tr>
          <td>id</td>
          <td>От кого</td>
          <td>Кому</td>
          <td>Сумма</td>
          <td>Статус</td>
        </tr>
      </thead>
      <tbody>
        {data.map((transaction, idx) =>
          <tr key={idx}>
            <td>{transaction.id}</td>
            <td>{transaction.account_from_id}</td>
            <td>{transaction.account_to_id}</td>
            <td>{transaction.amount}</td>
            <td>{transaction.result}</td>
          </tr>
        )}
      </tbody>
    </Table>
  )
};
```

```
        render() {
          const { card } = this.state;
          return (
            <div>
              {card.id
               ? <div>
                   <div className="cardAccountInfo">
                    <div className="cardWrapper">
                     <span className="removeCard" onClick={() =>
this.onRemove(card)}>&times;</span>
                     <div
className="cardNumber">{splitNumber(card.number)}</div>
                     <div
className="cardDate">{moment(card.expiry_date).format('MM/YY')}</div>
                    </div>
                    <div className="accountInfo">
                     <div className="accountField">
                      <div className="accountFieldTitle">Баланс:</div>
                      <div
className="accountFieldContent">{card.account.amount}</div>
                     </div>
                    </div>
                   </div>

                   <div>
                    <h5>История транзакций</h5>
                    <Tabs id="transactions-tab">
```

```
            <Tab eventKey={1} title="Исходящие">
              {this.renderTable(card.account.transactions_created)}
            </Tab>
            <Tab eventKey={2} title="Входящие">
              {this.renderTable(card.account.transactions_received)}
            </Tab>
          </Tabs>
        </div>
      </div>
     : <div>Такой карты не существует</div>
    }
  </div>
 );
}
}

export default Card;
```

8) Код компоненти Claims

```
import React, { Component } from 'react';
import client from '../../helpers/ApiClient';
import { Table } from 'react-bootstrap';

export default class Claims extends Component {
 constructor(props) {
  super(props);
  this.state = {
```

```
      claims: {
        meta: {
          total: 0
        },
        resources: []
      }
    };
  }

  componentDidMount() {
    client.get('/claims')
      .then(response => this.setState({
        claims: response
      }));
  }

  render() {
    const { claims } = this.state;
    const table = claims.resources.length > 0 && <Table striped bordered
condensed hover>
      <thead>
      <tr>
        <td>id</td>
        <td>На кого создана</td>
        <td>Описание</td>
      </tr>
      </thead>
      <tbody>
```

```
        {claims.resources.map((claim, idx) =>
          <tr key={idx}>
            <td>{claim.id}</td>
            <td>{claim.user.first_name + " + claim.user.last_name}</td>
            <td>{claim.description}</td>
          </tr>
        )}
        </tbody>
      </Table>;
      return (
        <div>
          <h3>Мои жалобы</h3>
          { table }
          { claims.resources.length < 1 && <h4 className="text-center">Вы не
подали ни одной жалобы</h4> }
        </div>
      );
    }
  }
```

9) Код компоненти Settings

```
import React, { Component } from 'react';
import { Route, Switch } from 'react-router-dom';
import Tabs from '../../components/Tabs/Tabs';
import Cards from './Cards';
import Profile from './Profile';
```

```
import './styles.css';

class Settings extends Component {
 render() {
   const { history } = this.props;
   const tabs = [
     {
       label: 'Мои счета',
       link: '/settings/cards'
     },
     {
       label: 'Профиль',
       link: '/settings/profile'
     }
   ];
   return (
     <div>
       <h2>Настройки</h2>
       <div className="container">
         <Tabs tabs={tabs} history={history}/>
         <div className="settingsContent">
           <Switch>
             <Route path="/settings/cards" component={Cards}/>
             <Route path="/settings/profile" component={Profile}/>
           </Switch>
         </div>
       </div>
     </div>
```

```
  );
 }
}

export default Settings;
```

10) Код компоненти Profile

```
import React, { Component } from 'react';
import { reduxForm, Field } from 'redux-form';
import { Button } from 'react-bootstrap';
import './styles.css';
import { connect } from 'react-redux';
import client from '../../helpers/ApiClient';
import { setUser } from '../../reducers/auth';

class Profile extends Component {

  static Form = reduxForm({
    form: 'userForm',
    fields: ['first_name', 'last_name', 'birth_date', 'email']
  })(({handleSubmit}) => (
    <div className="userFormWrapper">
      <form onSubmit={handleSubmit}>
        <div className="form-group">
          <label >Имя</label>
          <Field
```

```
    name="first_name"
    className="form-control"
    component="input"
    type="text"
    placeholder="Имя"
   />
</div>
<div className="form-group">
  <label >Фамилия</label>
  <Field
    name="last_name"
    className="form-control"
    component="input"
    type="text"
    placeholder="Фамилия"
  />
</div>
<div className="form-group">
  <label >Дата рождения</label>
  <Field
    name="birth_date"
    className="form-control"
    component="input"
    type="date"
  />
</div>
<div className="form-group">
  <label >Email</label>
```

```
            <Field
              name="email"
              className="form-control"
              component="input"
              type="email"
              placeholder="Email"
            />
          </div>
          <div className="text-center">
            <Button type="submit" bsStyle="primary"> Сохранить </Button>
          </div>
        </form>
      </div>
    ));

    onSubmit = (data) => {
      const { user } = this.props;
      client.put(`/users/${user.id}`, { data: {user: data}})
        .then(response => this.props.setUser(response.resource));
    };

    render() {
      return (
        <div>
          <Profile.Form onSubmit={data => this.onSubmit(data)}
initialValues={this.props.user}/>
        </div>
      );
```

```
  }
}

export default connect(state => ({user: state.auth.user}), { setUser
})(Profile);
```

11) Код компоненти Cards

```
import React, { Component } from 'react';
import './styles.css';
import client from '../../helpers/ApiClient';
import { Button } from 'react-bootstrap';
import { connect } from 'react-redux';
import { removeCard, addCard } from '../../reducers/cards';
import AddCardModal from
'../../components/AddCardModal/AddCardModal';
import Card from '../../components/Card/Card';

class Cards extends Component {
  constructor(props) {
    super(props);
    this.state = {
      showModal: false
    };
  }

  onShowModal = () => {
    this.setState({
```

```
    showModal: true
  });
};

onCloseModal = () => {
  this.setState({
    showModal: false
  });
};

onCreateCard = (data) => {
  const { user } = this.props;
  client.post('cards', { data: { card: {...data, user_id: user.id } }})
    .then(response => {
      this.props.addCard(response.resource);
      this.onCloseModal();
    });
};


onRemove = (card) => {
  client.del(`cards/${card.id}`)
    .then(() => this.props.removeCard(card));
};

render() {
  const { cards } = this.props;
  const { showModal } = this.state;
```

```
        return (
          <div>
            <div className="text-right addCardBtnWrapper">
              <Button type="submit" bsStyle="primary"
onClick={this.onShowModal}> Добавить карту </Button>
            </div>
            {cards.map((card, idx) =>
              <Card key={idx} withLink card={card} onRemove={() =>
this.onRemove(card)}/>
            )}
            <AddCardModal isOpen={showModal}
onSubmit={this.onCreateCard} onClose={this.onCloseModal} />
          </div>
        );
      }
    }

    export default connect(state => ({
      user: state.auth.user,
      cards: state.cards
    }), { removeCard, addCard })(Cards);
```

12) Код компоненти Transactions

```
    import React, { Component } from 'react';
    import client from '../../helpers/ApiClient';
    import { Table, Button, Tabs, Tab } from 'react-bootstrap';
```

```
import CreateClaimModal from
'../../components/CreateClaimModal/CreateClaimModal';
import { get } from 'lodash';

export default class Transactions extends Component {
 constructor(props) {
  super(props);
  this.state = {
   transactions: {
    meta: {
     total: 0
    }
   },
   transaction: null
  };
 }

 componentDidMount() {
  client.get('/transactions')
   .then(response => this.setState({
    transactions: response
   }));
 }


 onShowModal = (transaction) => {
  this.setState({
   transaction: transaction
```

```
  });
};

onCloseModal = () => {
  this.setState({
    transaction: null
  });
};

onCreateClaim = (data) => {
  console.log(data);
  // const { transaction } = this.state;
  client.post('/claims', { data: {
    resource: {
      // account_ids: [transaction.]
    }
  }})
};

renderTable = (data) => {
  return (
    <Table striped bordered condensed hover>
      <thead>
      <tr>
        <td>id</td>
        <td>От кого</td>
        <td>Кому</td>
        <td>Сумма</td>
```

```
            <td>Статус</td>
            <td>Жалоба</td>
          </tr>
          </thead>
          <tbody>
          {data.map((transaction, idx) =>
            <tr key={idx}>
              <td>{transaction.id}</td>
              <td>{transaction.account_from_id}</td>
              <td>{transaction.account_to_id}</td>
              <td>{transaction.amount}</td>
              <td>{transaction.result}</td>
              <td>
                <div className="text-center">
                  <Button bsStyle="danger" onClick={() =>
this.onShowModal(transaction)}> Подать жалобу </Button>
                </div>
              </td>
            </tr>
          )}
          </tbody>
        </Table>
      )
    };

    render() {
      const { transactions, transaction } = this.state;
      return (
```

```
            <div>
              <h3>Мои тразакции({transactions.meta.total})</h3>
              <Tabs id="transactions-tab">
                <Tab eventKey={1} title="Исходящие">
                  {get(transactions, 'resources.created') &&
this.renderTable(transactions.resources.created)}
                </Tab>
                <Tab eventKey={2} title="Входящие">
                   {get(transactions, 'resources.received') &&
this.renderTable(transactions.resources.received)}
                </Tab>
              </Tabs>
              <CreateClaimModal isOpen={transaction} transaction={transaction}
onSubmit={this.onCreateClaim} onClose={this.onCloseModal}/>
            </div>
          );
        }
       }
```

13) Код компоненти AddCardModal

```
import React, { Component } from 'react';
import Modal from 'react-modal';
import { reduxForm, Field } from 'redux-form';
import { Button } from 'react-bootstrap';

const styles = {
 overlay : {
```

```
  position          : 'fixed',
  top               : 0,
  left              : 0,
  right             : 0,
  bottom            : 0,
  backgroundColor   : 'rgba(255, 255, 255, 0.85)'
 },
 content : {
  position                  : 'absolute',
  top                       : '40px',
  left                      : '50%',
  transform                 : 'translateX(-50%)',
  width                     : '400px',
  right                     : '40px',
  bottom                    : '40px',
  border                    : '1px solid #ccc',
  background                : '#fff',
  overflow                  : 'auto',
  WebkitOverflowScrolling   : 'touch',
  height                    : '400px',
  outline                   : 'none',
  padding                   : '20px'

 }
};

class AddCardModal extends Component {
```

```
static Form = reduxForm({
  form: 'addCardForm',
  fields: ['number', 'expiry_date', 'cvv2']
})(({handleSubmit}) => (
  <div>
    <form onSubmit={handleSubmit}>
      <div className="form-group">
        <label >Номер</label>
        <Field
          name="number"
          className="form-control"
          component="input"
          type="text"
          placeholder="Номер"
        />
      </div>
      <div className="form-group">
        <label >CVV2</label>
        <Field
          name="cvv2"
          className="form-control"
          component="input"
          type="text"
          placeholder="cvv2"
        />
      </div>
      <div className="form-group">
        <label >Expiry date</label>
```

```
        <Field
          name="expiry_date"
          className="form-control"
          component="input"
          type="date"
        />
      </div>
      <div className="text-center">
        <Button type="submit" bsStyle="primary"> Добавить </Button>
      </div>
    </form>
  </div>
));

render() {
  const { isOpen, onSubmit, onClose } = this.props;
  return (
    <Modal isOpen={isOpen} style={styles} onRequestClose={onClose}>
      <h4>Добавить карту</h4>
      <AddCardModal.Form onSubmit={onSubmit}/>
    </Modal>
  )
 }
}

export default AddCardModal;
```

14) Код компоненти Card

```
import React, { Component } from 'react';
import { splitNumber } from '../../helpers/card';
import moment from 'moment';
import { get } from 'lodash';
import { Link } from 'react-router-dom';

export default class CardPreview extends Component {
 render() {
   const { card, onRemove, withLink } = this.props;
   return (
    <div className="cardAccountInfo">
     <div className="cardWrapper">
      { onRemove && <span className="removeCard"
onClick={onRemove}>&times;</span>}
      <div className="cardNumber">{splitNumber(card.number)}</div>
      <div
className="cardDate">{moment(card.expiry_date).format('MM/YY')}</div>
     </div>
     <div className="accountInfo">
      <div className="accountField">
       <div className="accountFieldTitle">Баланс:</div>
       <div className="accountFieldContent">{get(card,
'account.ammount') || 0}</div>
      </div>
      {withLink &&
       <div className="accountField">
        <Link to={`/cards/${card.id}`}>Транзакции</Link>
```

```
          </div>
        }
      </div>
    </div>
  )
 }
}
```

15) Код компоненти CreateClaimModal

```
import React, { Component } from 'react';
import Modal from 'react-modal';
import { reduxForm, Field } from 'redux-form';
import { Button } from 'react-bootstrap';

const styles = {
 overlay : {
  position          : 'fixed',
  top               : 0,
  left              : 0,
  right             : 0,
  bottom            : 0,
  backgroundColor   : 'rgba(255, 255, 255, 0.85)'
 },
 content : {
  position                  : 'absolute',
  top                       : '40px',
  width                     : '400px',
```

```
  left                    : '50%',
  transform               : 'translateX(-50%)',
  right                   : '40px',
  bottom                  : '40px',
  border                  : '1px solid #ccc',
  background              : '#fff',
  overflow                : 'auto',
  WebkitOverflowScrolling : 'touch',
  height                  : '380px',
  outline                 : 'none',
  padding                 : '20px'

 }
};

class CreateClaimModal extends Component {

 static Form = reduxForm({
  form: 'addClaimForm',
  fields: ['description']
 })(({handleSubmit}) => (
  <div>
    <form onSubmit={handleSubmit}>
      <div className="form-group">
        <label >Описание жалобы</label>
        <Field
          name="description"
          className="form-control"
```

```
          component="textarea"
          placeholder="Введите текст жалобы"
        />
      </div>
      <div className="text-center">
        <Button type="submit" bsStyle="primary"> Отправить </Button>
      </div>
    </form>
  </div>
));

 render() {
  const { isOpen, onSubmit, onClose } = this.props;
  return (
    <Modal isOpen={isOpen} style={styles} onRequestClose={onClose}>
      <h4>Создать жалобу</h4>
      <CreateClaimModal.Form onSubmit={onSubmit}/>
    </Modal>
  )
 }
}

export default CreateClaimModal;
```

16) Код компоненти CreateTransactionModal

```
import React, { Component } from 'react';
import Modal from 'react-modal';
```

```
import { reduxForm, Field } from 'redux-form';
import { Button } from 'react-bootstrap';
import { connect } from 'react-redux';

const styles = {
 overlay : {
   position         : 'fixed',
   top              : 0,
   left             : 0,
   right            : 0,
   bottom           : 0,
   backgroundColor   : 'rgba(255, 255, 255, 0.85)'
 },
 content : {
   position                 : 'absolute',
   top                      : '40px',
   width                    : '400px',
   left                     : '50%',
   transform                : 'translateX(-50%)',
   right                    : '40px',
   bottom                   : '40px',
   border                   : '1px solid #ccc',
   background               : '#fff',
   overflow                 : 'auto',
   WebkitOverflowScrolling    : 'touch',
   height                   : '400px',
   outline                  : 'none',
   padding                  : '20px'
```

```
 }
};

class CreateTransactionModal extends Component {

 static Form = reduxForm({
  form: 'addCardForm',
  fields: ['card_id', 'card_to_number', 'amount']
 })(({handleSubmit, cards}) => (
  <div>
   <form onSubmit={handleSubmit}>
    <div className="form-group">
     <label >Выберите карту</label>
     <Field
      name="card_id"
      className="form-control"
      component="select"
      type="text"
      placeholder="Номер"
     >
      {cards.map((card, idx) => <option key={idx}
value={card.id}>{card.number}</option>)}
     </Field>
    </div>
    <div className="form-group">
     <label >Сумма</label>
     <Field
```

```
          name="amount"
          className="form-control"
          component="input"
          type="text"
        />
      </div>
      <div className="form-group">
        <label >Номер карты получателя</label>
        <Field
          name="card_to_number"
          className="form-control"
          component="input"
          placeholder="4242424242424242"
          type="text"
        />
      </div>
      <div className="text-center">
        <Button type="submit" bsStyle="primary"> Создать </Button>
      </div>
    </form>
  </div>
));

render() {
  const { isOpen, onSubmit, onClose, cards } = this.props;
  return (
    <Modal isOpen={isOpen} style={styles} onRequestClose={onClose}>
      <h4>Создать транзакцию</h4>
```

```
            <CreateTransactionModal.Form cards={cards}
onSubmit={onSubmit}/>
          </Modal>
        )
       }
      }

      export default connect(state => ({cards:
state.cards}))(CreateTransactionModal);
```

17) Код компоненти Tabs

```
      import React, { Component } from 'react';
      import { Link } from 'react-router-dom';

      class Tabs extends Component {
       render() {
        const { tabs, history } = this.props;
        return (
          <div className="tabsWrapper">
            <ul className="nav nav-tabs">
              {tabs.map((tab, idx) =>
                <li key={idx}
className={history.location.pathname.includes(tab.link) ? 'active' : ''}>
                  <Link to={tab.link}>{tab.label}</Link>
                </li>
              )}
            </ul>
```

```
      </div>
    );
  }
}

export default Tabs;
```

18) Код ApiClient

```
import superagent from 'superagent';
import { setAuthDataToStore } from '../reducers/auth';
import { getAuthData, headers, clearAuthData } from './authData';
import includes from 'lodash/includes';
import keys from 'lodash/keys';

const methods = ['get', 'post', 'put', 'patch', 'del'];

function formatUrl(path) {
 const adjustedPath = path[0] !== '/' ? '/' + path : path;
 return `/api${adjustedPath}`;
}

class ApiClient {
 constructor() {
   methods.forEach((method) =>
     this[method] = (path, { params, data, attach } = {}) => new
Promise((resolve, reject) => {
       const request = superagent[method](formatUrl(path));
```

```
const authDataSource = getAuthData();

if (includes(keys(authDataSource), ...headers)) {
  headers.forEach((header) => {
    request.set(header, authDataSource[header]);
  });
  request.set('token-type', 'Bearer');
}

if (params) {
  request.query(params);
}

if (attach) {
  for (const [fieldName, file] of Object.entries(attach)) {
    request.attach(fieldName, file);
  }
}

if (data) {
  request.send(data);
}
request.end((error, response) => {
  if (error) {
    if (error.status === 401) {
      clearAuthData();
    }
    reject(response.body || error);
```

```
        } else {
          if ((/auth/).test(path) && response.headers['access-token']) {
            this.store.dispatch(setAuthDataToStore(response.headers));
          }
          return resolve(response.body);
        }
      });
    }));
 }
 setStore = (store) => {
   this.store = store;
 };
}

export default new ApiClient();
```

19) Код AuthData

```
import cookies from 'browser-cookies';
import { isEmpty, pickBy } from 'lodash';

export const headers = ['uid', 'access-token', 'client', 'expiry'];

export function setAuthData(data) {
 if (data) {
   headers.forEach((key) => {
     cookies.set(key, data[key].toString(), { expires: 14 });
   });
 }
```

```
}

export function getAuthData() {
 const data = {};
 headers.forEach((key) => {
   data[key] = cookies.get(key);
 });
 return data;
}

export function clearAuthData() {
 headers.forEach((key) => cookies.erase(key));
}

export function prepareAuthData(data) {
 const preparedData = {};
 headers.forEach(header => {
   preparedData[header] = data[header];
 });
 return !isEmpty(pickBy(preparedData)) ? preparedData : false;
}
```

20) Код store

```
import { createStore, combineReducers } from 'redux';
import { reducer as formReducer } from 'redux-form';
import auth from './auth';
import cards from './cards';
```

```
export default createStore(
 combineReducers({
   auth,
   cards,
   form: formReducer
 }),
 process.env.NODE_ENV === 'development'
 && window.__REDUX_DEVTOOLS_EXTENSION__
 && window.__REDUX_DEVTOOLS_EXTENSION__(),
);
```

21) Код редьюсера card

```
import { without } from 'lodash';
const SET_CARDS = 'cards/SET_CARDS';
const ADD_CARD = 'cards/ADD_CARD';
const REMOVE_CARD = 'cards/REMOVE_CARD';
const initialState = [];

export default function reducer(state = initialState, action) {
 switch (action.type) {
   case SET_CARDS:
     return [...state, ...action.payload];
   case ADD_CARD:
     return [action.payload, ...state];
   case REMOVE_CARD:
```

```
    return without(state, action.payload);
  default:
    return state;
 }
}

export function setCards(resources) {
 return {
  type: SET_CARDS,
  payload: resources
 };
}

export function removeCard(card) {
 return {
  type: REMOVE_CARD,
  payload: card
 };
}

export function addCard(card) {
 return {
  type: ADD_CARD,
  payload: card
 };
}
```

22) Код редьюсера auth

```
import { clearAuthData, prepareAuthData, setAuthData } from
'../helpers/authData';
const SET_USER = 'auth/SET_USER';
const LOGOUT = 'auth/LOGOUT';
const SET_AUTH_DATA_TO_STORE =
'auth/SET_AUTH_DATA_TO_STORE';

const initialState = {
 user: {}
};

export default function reducer(state = initialState, action) {
 switch (action.type) {
  case SET_USER:
   return {
    ...state,
    user: action.payload
   };
  case SET_AUTH_DATA_TO_STORE:
   setAuthData(action.payload);
   return {
    ...state,
    auth_data: action.payload
   };
  case LOGOUT:
   return {
    ...state,
```

```
      user: null
    };
  default:
    return state;
 }
}

export function setUser(resource) {
 return {
  type: SET_USER,
  payload: resource
 };
}

export function setAuthDataToStore(data) {
 return {
  type: SET_AUTH_DATA_TO_STORE,
  payload: prepareAuthData(data)
 };
}

export function logout() {
 clearAuthData();
 return {
  type: LOGOUT
 };
}
```

23) Код компоненти Routes

```
import React from 'react';
import Home from './containers/Home/Home';
import Settings from './containers/Settings/Settings';
import Card from './containers/Card/Card';
import Transactions from './containers/Transactions/Transactions';
import Claims from './containers/Claims/Claims';
import { Route, Switch } from 'react-router-dom';

export default () => (
 <Switch>
   <Route exact path='/' component={Home}/>
   <Route path='/settings' component={Settings}/>
   <Route path='/transactions' component={Transactions}/>
   <Route path='/claims' component={Claims}/>
   <Route path='/cards/:id' component={Card}/>
 </Switch>
);
```